

**THÈSE / UNIVERSITÉ DE RENNES 1**  
*sous le sceau de l'Université Bretagne Loire*

pour le grade de  
**DOCTEUR DE L'UNIVERSITÉ DE RENNES 1**

*Mention : informatique*

**Ecole doctorale MathSTIC**

présentée par

**Raphaël Bost**

préparée à l'Institut de Recherche en Informatique et Systèmes  
Aléatoires (IRISA), UMR 6074

---

**Algorithmes de  
recherche sur bases  
de données chiffrées**

**Thèse soutenue à Rennes  
le 8 Janvier 2018**

devant le jury composé de :

**Seny KAMARA**

Brown University / rapporteur

**Fabien LAGUILLAUMIE**

Université de Lyon 1 / rapporteur

**Kenneth G. PATERSON**

Royal Holloway, University of London / rapporteur

**David GROSS-AMBLARD**

Université de Rennes 1 / examinateur

**Sarah MEIKLEJOHN**

University College London / examinatrice

**Olga OHRIMENKO**

Microsoft Research Cambridge / examinatrice

**Pierre-Alain FOUQUE**

Université de Rennes 1 / directeur de thèse

**David POINTCHEVAL**

Ecole Normale Supérieure / directeur de thèse



# **Searchable Encryption**

**New Constructions of Encrypted Databases**

Raphael BOST

Supervisors: Pierre-Alain FOUQUE & David POINTCHEVAL



# Remerciements

J'ai connu des expériences plus reposantes que ces trois années de thèse. Par contre, rien d'aussi intéressant. Je dois cela à Pierre-Alain et à David, qui m'auront accompagné tout du long et envers qui je suis très reconnaissant. Pierre-Alain, merci pour ton enthousiasme, ta gentillesse, ta disponibilité et ta patience, tous sans faille — sans toi, je n'aurais sans doute pas profité d'une aussi grande indépendance dans mes recherches. David, merci pour le temps que tu as pu me consacrer lors de mes visites impromptues à l'ENS, tes remarques toujours justes et constructives, et bien évidemment, le sujet de thèse qui, comme le montre ce manuscrit, m'a passionné. Merci à vous deux pour votre encadrement qui m'a beaucoup appris, non seulement scientifiquement, mais aussi humainement.

I deeply thank the reviewers of this thesis, Fabien Laguillaumie, Seny Kamara, and Kenny Paterson for the time they spent reading it, and helping me to improve it, and in general for their interest in my work. I also thank David Gross-Amblard, Sarah Meiklejohn, and Olga Ohrimenko for accepting to be part of my jury.

I would have done a lot less interesting things without the great co-authors and collaborators I had, and with whom I had very enriching experiences. All this would have been a lot more difficult without Raluca, Stephen, Shafi, Mario, Olivier, Brice and Olya. I think we solved great and very interesting problems, and hope I will have the occasion to do this again with you.

When I graduated from Ecole Polytechnique, I did not have in mind to pursue a Ph.D. I had joined it to be an engineer, I had done some engineering studies, so I did not really wanted to get in some fancy, mostly theoretical and not applied research. Gilles Dowek and Pierre Néron, while I was a research intern in their team, started to make me change my mind. Back then, I learned that research can be very exciting, and even theoretical results can have huge impacts in practice.

What definitely changed my mind was the time I spent at Brown. Thanks to Anna Lysyanskaya, I discovered what cryptography really was, and was introduced to the amazing world of provable cryptography. Anna, your classes truly were a revelation to me, and most probably changed my life. Also, I would not have worked on a topic at the crossing of applied cryptography, security and systems without Olya, who spent quite some time explaining me her work on ORAM. Оля, ты меня вдохновляешь. Большое спасибо!

Les nombreux Mercredis passés à l'IRISA, dans l'équipe EMSEC, auront donc porté leurs fruits. Ce furent des journées bien remplies, de discussions scientifiques certes, mais aussi de débats peu constructifs et d'humour approximatif, en particulier avec les deux générations successives du bureau F413. Je remercie donc Brice et Pierre pour les échanges extra-cryptographiques portant sur le nommage de nouveaux algorithmes, le design du logo de l'équipe, celui de ma magnifique affiche de thèse, ou encore les chats — je leur suis légèrement moins reconnaissant d'avoir organisé leurs vacances en Nouvelle-Zélande devant moi — ainsi que Pierre d'avoir toléré de partager sa place sur le bureau avec moi. Il a été au moins aussi difficile de travailler avec la seconde génération de thésards dans le bureau, entre Florent — toujours à la pointe du progrès, grâce à aux recompilations

incessantes de son noyau — Alban — notre spécialiste stats et Python, le pauvre — et Wojciech — whose plants are the only normal life beings in the office. Je n’oublie évidemment pas les autres doctorants et post-docs qui ont eux aussi participé à cette bonne ambiance, Alexandre, Angèle, Baptiste, Benjamin, Céline, Chen, Claire, Cristina, Cyrille, Guillaume, Jean-Christophe, Mohammed, Pauline, Thomas et Vincent. Les membres permanents de l’équipe ne sont pas en reste, et tout particulièrement Adeline et Benoit, qui n’ont eu de cesse de partager avec moi leur bonne humeur et leur motivation, ainsi que Barbara, Clémentine, Gildas, Patrick et Stéphanie.

Je n’ai malheureusement pas pu passer autant de temps que j’aurais voulu à discuter avec les membres de l’équipe crypto de l’ENS, mais je tiens les remercier pour l’accueil que j’y ai toujours reçu : Adrian, Alain, Dahmun, Damien, Fabrice, Florian, Geoffroy, Georg, Hoeteck, Louiza, Mario, Michel, Michele, Pierre-Alain, Pierrick, Romain, Rafael, Sonia et Thomas.

Je n’aurais pas pu faire cette thèse sans la possibilité qui ma été offerte de faire de la recherche en parallèle de mon poste à la DGA, grâce à l’implication de Gwenaëlle et de Laurent. J’espère que les résultats de cette thèse sauront servir d’explication suffisante à mes pérégrinations hebdomadaires à l’université et à l’utilité de la fonction technique “pure”. En outre, mon travail pour la DGA m’a permis de découvrir nombre d’aspects pratiques de la cryptographie que je n’avais pas soupçonnés jusqu’alors. Merci donc à Didier de m’avoir permis d’entrer dans cette équipe, ainsi qu’à tous mes collègues du laboratoire, Céline, Clément, Cyrille, Gaël, Gwezheneg, Jean-Gabriel, Julien, Olivier, Pierre, Reynald, Stéphanie, Valérie et Victor ; du département, Yannick (cité en premier pour qu’il ne se plaigne pas), Arnaud, Benoit, Claire, David, Didier, Hugues, Jérôme, Laurent, Laurent, Laurent, Marec, Mathieu, Nabil, Olivier, Oswald, Stéphane et Thierry ; sans oublier Anne-Marie, David, Dominique, Dominique, Franck, François, Gurvan, Luc, Marie-Charlotte, Olivier, Philippe, Sofia, Stéphane, Sylvain et Thomas, ni mon stagiaire Derrick.

Comment ne pas penser aux camarades du Café Cortina, qui ont apporté leur soutien moral les Jeudis soirs de ces trois dernières années. Rennes ne serait pas tout à fait pareil sans l’hospitalité et la gentillesse d’Anne, de Thierry et d’Aline, sans les aventures asiatiques de Yuning, de Nabil, de Gregory et de Hugues, sans le chasuble jaune et la casquette de Gaby, sans les photos de vacances d’Anne, sans les madeleines de Cyrille, sans l’humour décapant de Brigitte et de Marie, sans l’us en feu de Nicolas, sans les verres à finir d’Adeline, sans les “c’est chaud, quoi” de Victor, sans les deux-pintes-mais-pas-plus de Jordan. Jordan, justement, merci pour ces deux années de collocation, pendant lesquelles on se sera bien amusé. J’en profite pour adresser tout mon soutien à Marie, pour les futures baies vitrées brisées ou les échecs devant le lave-vaisselle.

Je ne peux pas terminer ces remerciements sans penser à ma famille. A Pierre, qui commet la même malheureuse erreur de se lancer lui aussi dans une thèse, mais qui, j’en suis certain, réussira brillamment. A mes parents, bien sûr, qui m’auront donné goût à l’informatique en m’installant devant un émulateur DOS avec un compilateur de BASIC, à Maman malgré ses cours de maths particuliers traumatisants (mais qui ont probablement eu un certain succès), à Papa qui a toujours pris de son temps pour m’aider et répondre à mes questions (et aussi pour répondre à d’autres questions que je ne me posais pas et ainsi terroriser certains de mes camarades).

Enfin, merci Adeline pour ta patience avec moi, pour avoir supporté ma mauvaise humeur régulière, pour ton soutien lors des soirées studieuses et des journées au moral amputé. Merci d’avoir été aussi attentionnée et compréhensive. Tu es pour beaucoup dans le succès de cette thèse.









# Résumé en français



COMMENT PEUT-ON CHIFFRER UNE BASE DE DONNÉES tout en conservant un certain nombre de fonctionnalités, et en particulier la possibilité d'effectuer efficacement des recherches sur le contenu de cette base ? C'est à cette question que nous allons tenter de répondre dans cette thèse. Si elle semble simple, nous allons voir qu'en pratique, trouver une solution à ce problème est affaire de compromis entre sécurité et performance.

Ce chapitre va introduire cette problématique d'algorithmes de recherche sur des bases de données chiffrées puis résumer les solutions que j'ai développées durant mon doctorat. Enfin, je donnerai un aperçu rapide de deux autres travaux portant sur des sujets différents (mais qui traitent toujours de cryptographie) et qui ne seront pas détaillés dans cette thèse.

## Sommaire

---

<b>Chiffrement de base de données</b> . . . . .	<b>vi</b>
Pourquoi chiffrer des bases de données ? . . . . .	vi
Problématique du chiffrement des bases de données . . . . .	vii
<b>Travaux sur les algorithmes de recherche sur bases de données chiffrées</b> . . . . .	<b>vii</b>
Confidentialité persistante des algorithmes de recherche sur bases de données chiffrées . . . . .	viii
Confidentialité future des algorithmes de recherche sur bases de données chiffrées	ix
Algorithmes vérifiables de recherche . . . . .	ix
Attaques par abus de fuite . . . . .	x
<b>Autres travaux</b> . . . . .	<b>x</b>
Classification de données chiffrés par un algorithme d'apprentissage . . . . .	xi
Attaque sur le mode de chiffrement authentifié OTR . . . . .	xi

---

## Chiffrement de base de données

### Pourquoi chiffrer des bases de données ?

Au cours des dernières décennies, quantité de données a été progressivement externalisée depuis leurs possesseurs vers des fournisseurs de service, allant de la messagerie électronique, jusqu'aux informations sur des comptes bancaires, des consommateurs ou encore des photos personnelles. De plus en plus d'entreprises décident d'utiliser des plates-formes collaboratives dans leur travail quotidien. De même, certaines sociétés ne veulent plus avoir à gérer une infrastructure locale pour stocker la quantité sans cesse croissante d'information à laquelle ils ont à faire. Des fournisseurs tels que Google, Amazon, Apple, Microsoft, IBM, Dropbox, ou beaucoup d'autres, offrent une large gamme de services, tels que messagerie électronique, hébergement de fichiers, édition collaborative de documents ou encore de sauvegarde appelés de façon générique 'le Cloud'. Ces services sont très séduisants pour les sociétés privées, pour le grand public et voire pour le secteur public, car ils permettent à leurs utilisateurs de "se concentrer sur les données plutôt que sur l'infrastructure", et ce à un tarif très compétitif et avec de bonnes garanties de disponibilité et de sûreté.

Dans le même temps, la protection des données est devenue une question importante pour nombre d'entreprises travaillant sur des données sensibles. La première raison à cela est le développement d'une législation obligeant ces entreprises à sécuriser leurs données, en particulier lorsqu'il s'agit d'informations personnelles. Pour donner des exemples de telles lois, nous pouvons citer le Titre II de la loi HIPPA (*Health Insurance Portability and Accountability Act*) de 1996 pour le secteur de la santé et la loi Sarbanes-Oxley pour le secteur financier aux États-Unis d'Amérique, et pour l'Union Européenne, le *Règlement général sur la protection des données*, qui remplace la *Directive sur la protection des données*. Ces lois cherchent à prévenir la révélation ou l'accès, illégal ou non autorisé, aux données sensibles. Dans la plupart des cas, cela implique l'utilisation de chiffrement pour les données stockées, afin d'assurer leur confidentialité, mais aussi l'utilisation de mécanismes cryptographiques – authentification des utilisateurs, algorithmes d'intégrité – empêchant la modification non détectée de données. Il est important de mentionner qu'avec ces réglementations, l'externalisation du stockage d'informations médicales ou bancaires sans avoir recours à des protocoles cryptographiques de contrôle d'accès est rendu simplement illégal, et empêche beaucoup de fournisseurs de services sur le Cloud d'avoir accès à des parts de marché importantes.

Les nouvelles contraintes législatives ne sont pas les seules raisons motivant le développement de bases de données chiffrées. En effet, à la suite des révélations de Snowden de 2013 portant sur le programme d'interceptions électroniques de la National Security Agency des États-Unis d'Amérique – aidée en cela d'agences britanniques, australiennes ou allemandes – beaucoup d'utilisateurs de services sur le Cloud, et plus généralement des utilisateurs d'internet, aussi bien des individus que des entreprises, ont été convaincus de l'importance de chiffrer leurs communications et leurs données afin de protéger leur vie privée ou leurs activités professionnelles. En particulier, nous avons assisté à une perte de confiance importante en les fournisseurs de services, car ils peuvent être forcés – par un mandat – à révéler des clés de chiffrement protégeant les données qu'ils stockent.

Dans ce contexte, la protection des seules communications entre l'utilisateur et le fournisseur de services n'est pas suffisante, car le fournisseur continuera à voir les données que le client a envoyées en clair. Ainsi, les services modernes de messagerie instantanée tels que Signal ou WhatsApp mettent en œuvre du chiffrement "de bout en bout" qui permet aux parties qui communiquent de s'assurer qu'elles seules peuvent lire les messages échangés.

Pour les bases de données, nous voudrions de même éviter que le serveur puisse inférer des informations, que cela soit sur les données elles-mêmes ou sur les accès faits par le client. Il est aussi

essentiel que les constructions de chiffrement de bases de données utilisent des outils cryptographiques courants et que leur sécurité puisse être prouvée, afin que leurs utilisateurs aient confiance en de cette sécurité. Et, au delà d'être sûrs, ces systèmes doivent aussi être utilisables en pratique ; c'est-à-dire qu'ils doivent être facile à utiliser, avec un déploiement simple et des fonctionnalités similaires à leurs équivalents non sécurisés, que leurs performances doivent être compétitives avec celles d'une base de données non chiffrée, et enfin qu'elles puissent passer à l'échelle et supporter de très grandes bases (jusqu'à plusieurs téraoctets de données). Sans tout cela, les systèmes sécurisés ne seront jamais choisis en faveur des systèmes non-chiffrés : Signal et WhatsApp ont beaucoup plus de succès que PGP car ces deux applications sont très faciles d'utilisation, à l'opposé de la messagerie électronique chiffrée avec PGP. Nous verrons dans cette thèse qu'arriver à concilier sécurité et efficacité est déjà difficile pour des fonctionnalités simples (et dans certains cas même impossible).

En particulier, nous allons nous concentrer sur le chiffrement de bases de données supportant des requêtes de recherche portant sur un seul mot-clé, appelé dans la littérature anglophone "searchable encryption" (chiffrement recherachable).

## Problématique du chiffrement des bases de données

Afin de pouvoir traiter correctement les algorithmes de recherche sur des bases de données chiffrées, il nous faut introduire une définition de sécurité permettant de formellement quantifier leur sécurité, c'est-à-dire la confidentialité des données et des requêtes. Malheureusement, nous verrons qu'il est impossible de construire un schéma qui soit à la fois parfaitement sûr — cachant toutes les informations à propos des données et des requêtes, à l'exception peut-être de la taille de la base et du nombre de résultats — et efficace. Des définitions de sécurité prenant en compte la possibilité d'une fuite d'information ont donc été conçues. Elles sont paramétrées par une *fonction de fuite* qui justement modélise ce que le serveur peut apprendre : lorsque les définitions de sécurité ainsi paramétrées sont satisfaites, le serveur ne peut pas apprendre plus d'information que ce que lui révèle la fonction de fuite.

Ainsi, l'enjeu est de réduire le plus possible les fuites éventuelles tout en offrant les performances les meilleures possibles et des fonctionnalités avancées. Nous verrons dans les chapitres 3 et 4 que, malheureusement, il faudra faire un compromis, parfois important, entre sécurité et performance et que la sécurité a un coût incompressible : il n'est pas possible de construire des bases de données chiffrées avec un niveau de sécurité acceptable et dont les performances, même asymptotiques, sont comparables avec celles d'une base non chiffrée. En effet, certaines fuites rendent les constructions (très) facilement attaquable et éviter ces fuites nécessite une certaine complexité du schéma de chiffrement.

Les travaux de cette thèse visent à mieux appréhender les compromis entre sécurité et performance, à en trouver des optima, à comprendre ce qu'implique la fuite de certaines informations et plus généralement à trouver une solution sûre et utilisable au problème des bases de données chiffrées supportant des requêtes de recherche. Ils sont décrits plus en détail dans la section suivante.

## Travaux sur les algorithmes de recherche sur bases de données chiffrées

Je présente ici rapidement les travaux portant sur les bases de données chiffrées effectués durant mon doctorat. Ils seront développés *in extenso* dans la suite de la thèse.

## Confidentialité persistante des algorithmes de recherche sur bases de données chiffrées

La possibilité de modifier une base de données chiffrée est une fonctionnalité essentielle pour rendre utilisable de telles constructions : les bases de données statiques ne recouvrent qu'une faible partie des cas d'emploi de bases de données. Malheureusement, comme c'est souvent le cas en cryptographie, ajouter de nouvelles fonctionnalités entraîne l'arrivée de nouvelles vulnérabilités et surtout de nouveaux moyens d'attaques. En effet, il est compliqué pour un adversaire d'influencer *a priori* l'ensemble des données à chiffrer d'une base de données statique, alors que dans le cas de bases dynamiques, il peut pousser l'utilisateur à insérer des documents qui peuvent l'aider à récupérer des informations sur la base elle-même ou sur des requêtes, passées ou futures.

Ainsi, des attaques par injection de fichier ont été mises au point et permettent de casser la confidentialité des requêtes de recherche — c'est-à-dire de retrouver quel mot-clé a été utilisé pour la recherche. Il existe notamment des versions adaptatives de ces attaques, qui permettent de retrouver le mot-clé d'une requête passée avec une très grande efficacité. Ces attaques adaptatives utilisent le fait que, lors de l'insertion d'un nouveau document dans la base de données, le serveur peut apprendre si ce nouveau document correspond à une requête précédente. Grâce à cette information, l'attaquant peut effectuer une variante d'une recherche dichotomique et retrouver le mot-clé de la recherche ciblée.

Il est donc essentiel que les schémas de chiffrement de base de données dynamiques ne révèlent pas cette information. On dit alors qu'ils satisfont la propriété de *confidentialité persistante* (*forward privacy* en Anglais).

Avant mon travail, la seule construction existante dans la littérature satisfaisant cette propriété avait été développée par Stefanov, Shi et Papamanthou [SPS14], qui avaient à cette occasion informellement introduit ce concept de confidentialité persistante. Cependant cette construction, bien qu'élégante, est complexe, inefficace et n'est pas utilisable en pratique pour de grandes bases de données (en particulier pour des questions de bande passante nécessaire aux mises à jour).

Dans [Bos16], j'ai formalisé la propriété de confidentialité persistante en donnant une définition de sécurité claire, puis ai proposé une première construction d'algorithme satisfaisant celle-ci,  $\Sigma\phi\phi\phi$ , très simple, asymptotiquement optimale en terme d'efficacité calculatoire et basée sur un objet cryptographique bien connu, les permutations à trappe. J'ai de plus démontré son aspect pratique en proposant une implémentation.

Malheureusement, l'utilisation de permutations à trappe, et donc de cryptographie asymétrique, rend  $\Sigma\phi\phi\phi$  en pratique moins efficace que certaines autres constructions qui ne sont pas confidentielles de manière persistante. Avec Brice Minaud et Olga Ohrimenko, nous avons résolu ce problème dans [BMO17] en remplaçant la permutation par un arbre de fonctions pseudo-aléatoire. Si notre schéma, Diana, est en théorie moins efficace que  $\Sigma\phi\phi\phi$ , il le surpasse en pratique car les opérations de cryptographie symétrique nécessaires à Diana sont beaucoup plus rapides que les opérations asymétriques nécessaires à  $\Sigma\phi\phi\phi$ . *In fine*, Diana est aussi efficace que des schémas n'étant pas confidentiels de façon persistante.

Un point à souligner est que Diana comme  $\Sigma\phi\phi\phi$  nécessitent de faire stocker des éléments par l'utilisateur (un compteur par mot-clé présent dans la base). Je montre dans cette thèse que cela est inévitable si on s'interdit d'augmenter sensiblement la complexité calculatoire du schéma : il y a une borne inférieure sur la complexité calculatoire du protocole de mise à jour des constructions à confidentialité persistante.

Ces idées sont approfondies dans le Chapitre 4.

## Confidentialité future des algorithmes de recherche sur bases de données chiffrées

Ainsi, avec la confidentialité persistante, nous pouvons nous assurer que les modifications de la base de données ne peuvent pas être reliées à des recherches précédentes. Une autre notion naturelle à étudier est celle de la *confidentialité future* du système de chiffrement : les requêtes de recherche ne devraient pas faire fuiter de l'information sur des entrées et des résultats supprimés. La confidentialité future est ainsi fortement liée au problème de la suppression sécurisée de données : nous ne voulons pas que le serveur puisse recouvrer des informations supprimées.

Dans les constructions existantes, les suppressions étaient gérées en utilisant une liste de révocation : les documents étaient marqués comme supprimés et le serveur éliminait ces documents de la liste de résultats. En particulier, le serveur avait encore accès aux informations supprimées, alors que, paradoxalement, le propriétaire des données, l'utilisateur, ne l'avait plus.

La confidentialité future avait été introduite de façon très informelle par Stefanov, Shi et Papamanthou dans [SPS14], en même temps donc que la confidentialité persistante, mais est restée grandement négligée jusqu'à notre travail avec Brice Minaud et Olga Ohrimenko [BMO17]. Dans ce papier, nous formulons une définition formelle de la confidentialité future, avec plusieurs variantes, et nous construisons quatre schémas de chiffrement de bases de données dynamiques, Fides, Moneta, Diana<sub>del</sub> et Janus qui satisfont ces définitions avec différents compromis d'efficacité. Ce sont les premières constructions avec confidentialité future (et persistante).

Certaines de ces constructions utilisent des outils cryptographiques permettant un contrôle d'accès fin aux données chiffrées, tels que le chiffrement et les fonctions pseudo-aléatoire poinçonnables. Ils permettent en effet d'empêcher le déchiffrement de certains messages ou l'évaluation de fonctions sur certaines entrées. Nos constructions utilisent notamment ces primitives en empêchant le déchiffrement des entrées de la base de données qui ont été supprimées.

Enfin, nous avons aussi effectué une évaluation pratique d'une de nos constructions la plus innovante, Janus, démontrant par là même que, malgré son intérêt théorique, il ne s'agit pas d'une construction pouvant passer à l'échelle.

Les définitions, constructions et résultats sont reproduits dans le Chapitre 5.

## Algorithmes vérifiables de recherche

Précédemment, nous avons supposé que le serveur était "honnête mais curieux", c'est-à-dire qu'il essayait de casser la confidentialité du contenu de la base de donnée ou des termes de recherche, mais qu'il respectait toujours le protocole de recherche tel que prescrit. Mais le serveur pourrait être "malicieux", tricher et faire en sorte que l'utilisateur ne reçoive qu'une partie des résultats de recherche, ou bien ajouter de faux résultats.

Pour lutter contre ce type d'attaque, l'utilisateur se doit donc de mettre en place des mécanismes de vérification des résultats de requêtes. Dans [BFP16], avec Pierre-Alain Fouque et David Pointcheval, nous étudions ces mécanismes.

Dans une première partie, nous démontrons que la vérification des résultats a un coût inhérent : il existe à nouveau une borne inférieure sur la complexité calculatoire des algorithmes de recherche sûrs contre des adversaires malicieux. Le temps d'exécution de l'algorithme de recherche ou de l'algorithme de mise à jour doivent croître logarithmiquement en le nombre de mots-clés.

Ensuite, nous proposons une solution générique à ce problème, ainsi que deux instanciations. La première se base sur une structure de donnée de dictionnaire vérifiable basée sur des arbres de Merkle, la seconde utilise des accumulateurs cryptographiques.

Enfin, nous corrigeons un point de [SPS14] qui décrivait comment rendre la construction de ce papier vérifiable et sûre contre les adversaires malicieux. En particulier, prouver la sécurité du schéma de [SPS14] dans le modèle proposé par cet article s'avère assez délicat.

Le Chapitre 6 décrit ces résultats en détail et explique aussi comment rendre  $\Sigma\phi\phi\phi\phi$ , Diana et Janus vérifiables.

## Attaques par abus de fuite

Ma dernière contribution porte sur les attaques par abus de fuite et leur interprétation. Une attaque par abus de fuite utilise uniquement la fonction de fuite pour casser la confidentialité du système, et non une éventuelle faille dans sa construction. Ainsi, ce sont des attaques très générales, qui peuvent avoir une grande portée, puisqu'elles visent des constructions différentes pourvu qu'elles aient la même fonction de fuite.

Or, autoriser à fuiter des informations est essentiel pour que nos algorithmes aient une efficacité raisonnable. Ainsi, bien comprendre la portée des informations révélées au serveur est essentiel pour estimer la sécurité d'un système de chiffrement de bases de données. Il se pourrait en effet que ces informations soient suffisantes pour rendre ce système non sûr.

Ainsi, supposons que le serveur ait eu connaissance de la base de données en clair (par exemple, par ce qu'elle est publique et que l'utilisateur ne veut protéger que les requêtes). Il est très probable que, pour un certain nombre de mots-clés, le nombre de documents retournés par une recherche sur ce mot détermine uniquement de dernier. Le serveur peut alors calculer une table faisant correspondre ces mots-clés et le nombre de résultats, permettant d'identifier facilement le terme d'une requête uniquement à partir du nombre de résultats de cette requête.

Il est étonnant de voir que, d'une part, nous avons des constructions dont la sécurité est démontrée dans un modèle éprouvé, et que, d'autre part, il soit relativement aisé de mettre au point des attaques contre ces constructions. Dans [BF17], avec Pierre-Alain Fouque, nous avons tenté de mieux comprendre ce paradoxe. Nous montrons formellement qu'en fait, ces attaques sortent du modèle de sécurité dans lequel ces systèmes sont prouvés sûrs.

Pour résoudre ce problème, nous proposons de nouvelles définitions de sécurité capturant ces attaques par abus de fuite. De plus, nous présentons une méthode permettant d'évaluer la sécurité des constructions existantes vis-à-vis de ces nouvelles définitions. En particulier, nous utilisons cette méthode pour construire des algorithmes de chiffrement de base de données que l'on peut montrer sûrs contre ce type d'attaques et nous l'appliquons au cas de l'attaque par fréquence décrite plus haut.

Cette étude est présentée dans le Chapitre 7.

## Autres travaux

Au cours de ces années de doctorat, j'ai publié deux autres papiers sur des sujets différents du chiffrement de bases de données. Le premier, issu d'un travail effectué au Massachusetts Institute of Technology en fin de Master, co-écrit avec Raluca Ada Popa, Stephen Tu et Shafi Goldwasser, porte sur la classification automatique de données chiffrées à l'aide d'un algorithme d'apprentissage automatique. Le second, en collaboration avec Olivier Sanders, porte sur une attaque sur un mode de chiffrement authentifié appelé OTR.

## Classification de données chiffrés par un algorithme d'apprentissage [BPTG15]

L'apprentissage automatique (*machine learning* en Anglais) est utilisé aujourd'hui dans de nombreux domaines, tels que la médecine, la détection de pourriels, la reconnaissance faciale ou les prédictions financières. Notamment, les algorithmes d'apprentissage et de classification travaillent sur des données hautement sensibles et il est important de garder confidentiel ces données.

Considérons la configuration classique de l'apprentissage supervisé : dans un premier temps, la phase d'apprentissage, l'algorithme d'apprentissage construit un modèle à partir de données déjà étiquetées et classifiées, puis, dans un second temps, l'algorithme de classification utilise ce modèle pour prédire, à l'aide du modèle, une prédiction pour l'étiquette d'un élément pas encore classifié. Il est ainsi important qu'à la fois le modèle et les données à classifier restent protégées : un potentiel client d'une banque ne désire pas nécessairement donner sa situation bancaire, un hôpital ne peut pas révéler de données sur ses patients, quand bien même elles ne seraient qu'agrégées.

Idéalement, dans le cas d'un hôpital offrant des services de pré-diagnostic à ses patients, on souhaiterait que les parties exécutent un protocole où le patient n'apprend que le résultat de ce diagnostique (et rien d'autre sur le modèle de prédiction construit par l'hôpital) et que l'hôpital n'apprenne rien sur les antécédents médicaux du patient.

Ce travail cherche à résoudre concrètement ce problème et propose plusieurs outils permettant de construire des protocoles de classification préservant le modèle et les données sur lesquels il est appliqué. Pour cela, nous avons identifié un certain nombre de "briques" utilisées fréquemment dans les algorithmes de classification et nous avons construit des algorithmes permettant de les utiliser de manière sûre, sans faire fuir d'information. Parmi elles, on peut trouver le calcul d'un produit scalaire, la comparaison de deux valeurs, le calcul d'un maximum, ...

Dans un second temps, nous composons ces briques de base pour construire des protocoles de classification sûrs, couvrant un nombre important d'algorithmes d'apprentissage automatique. Ainsi, nous présentons des protocoles pour des classifieurs linéaires, pour des classifieurs de Bayes naïfs ou encore pour des arbres de décision. La sécurité de ces protocoles repose sur une preuve de sécurité.

Enfin, j'ai implémenté ces briques et ces classifieurs. Nous avons donc pu montrer que nos constructions sont efficaces, prenant quelques millisecondes à quelques secondes pour classifier des données médicales.

## Attaque sur le mode de chiffrement authentifié OTR [BS16]

La sécurité des communications concerne non seulement la confidentialité des données échangées à travers le canal, mais aussi leur intégrité. Des solutions efficaces existent pour ces deux propriétés prises séparément et peuvent être combinées. Aussi, la combinaison d'un algorithme pour la confidentialité et d'un algorithme pour l'intégrité est souvent sous-optimal par rapport à l'utilisation d'un mécanisme unique.

Toutefois, le développement d'un algorithme de chiffrement authentifié permettant de résoudre ce problème de manière efficace est devenu un sujet majeur en cryptographie et donné naissance à la compétition CAESAR.<sup>1</sup> Dans le cadre de cette compétition, de nombreux algorithmes nouveaux ont été proposés et parmi eux, OTR (pour Offset Two Rounds) de Minematsu [Min14].

Le schéma OTR utilise sur un algorithme de chiffrement par blocs adaptable (en Anglais, Tweakable Bloc Cipher ou TBC), une primitive très puissante qui consiste en un algorithme de chiffrement par bloc classique auquel on rajoute un paramètre public supplémentaire (le tweak), qui peut être contrôlé par l'attaquant. Plus précisément, OTR chiffre des paires de blocs de messages en appliquant

<sup>1</sup>Competition for Authenticated Encryption : Security, Applicability, and Robustness

deux tours d'un réseau de Feistel dont les fonctions internes sont deux appels à l'algorithme de chiffrement par bloc adaptable avec deux paramètres différents.

Pour instancier l'algorithme de chiffrement, une des méthodes utilisées — et notamment par OTR — est appelée XEX\* (pour Xor-Encrypt-Xor) : un masque  $\Delta$  est dérivé du paramètre et est xoré en entrée (et éventuellement en sortie) du bloc de chiffrement. Plus précisément, les masques sont des chaînes de  $n$  bits, représentant un élément du corps  $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/P(X)$  où  $P$  est un polynôme primitif de degré  $n$  sur  $\mathbb{F}_2$ , et sont de la forme  $(X^i + a \cdot X + b) \delta$  ou  $(X^{i+1} + X^i + a \cdot X + b) \delta$  où  $i$  est le numéro du bloc de message à chiffrer et avec  $a, b \in \{0, 1\}$  et  $\delta \in \mathbb{F}_2[X]/P(X)$  généré (pseudo-)aléatoirement. Un élément essentiel pour la sécurité de cet algorithme est le fait qu'il n'y ait pas de collision entre les masques.

Or il se trouve que rien n'empêche d'avoir des collisions entre les polynômes de la forme  $X^i + a \cdot X + b$  ou  $X^{i+1} + X^i + a \cdot X + b$  pour des faibles valeurs de  $i$  (correspondant donc à des petits messages). Pour certaines valeurs de  $P$  utilisées communément, ces collisions sont même immédiates à trouver. Par exemple, pour  $n = 64$ , on utilise souvent  $P(X) = X^{64} + X^4 + X^3 + X + 1$ , on a une collision immédiate entre  $X^{64}$  et  $X^4 + X^3 + X + 1$  dans  $\mathbb{F}_2[X]/P(X)$ .

Dans ce papier, nous avons soulevé ce problème et montré que pour une grande partie des tailles de bloc  $n \leq 10\,000$ , on peut trouver des collisions entre les polynômes immédiatement. Puis nous nous sommes concentrés sur les tailles de bloc usuelles ( $n = 64$  et  $n = 128$ ). Si l'on peut trouver des collisions immédiates pour  $n = 64$  avec le polynôme primitif utilisé habituellement pour construire  $\mathbb{F}_{2^{64}}$ , ce n'est pas le cas pour  $n = 128$  et  $P(X) = X^{128} + X^7 + X^2 + X + 1$  utilisé en pratique pour définir  $\mathbb{F}_{2^{128}}$ . Nous avons donc étudié de manière plus approfondie ce cas et montré, par le calcul, qu'il n'y avait pas de collision possible entre les polynômes utilisés pour la génération des masques pour  $i < 2^{45}$  (c'est-à-dire pour des tailles de message raisonnables). Nous émettons aussi la conjecture, justifiée par des expériences, que les collisions ne devraient pas arriver pour  $i < 2^{60}$ .

Bien que l'existence de telles collisions invalide la preuve de sécurité d'OTR, cela ne nous donne pas une attaque sur l'algorithme de chiffrement. Dans une seconde partie de notre travail, nous montrons donc comment ces collisions peuvent être utilisées pour casser la confidentialité et/ou l'intégrité des données chiffrées.

Enfin, nous décrivons plusieurs méthodes possibles pour la construction des masques utilisées dans l'algorithme de chiffrement par blocs adaptable évitant ces collisions. Nous insistons ainsi sur le fait que notre travail ne remet pas en question la sécurité intrinsèque de l'algorithme OTR vu en tant qu'algorithme utilisant un chiffrement par blocs adaptable en boîte noire, mais que l'instanciation de ces derniers dans la version originale d'OTR devait être corrigée. Ainsi, suite à notre attaque, l'auteur a modifié la génération des masques dans la dernière version d'OTR soumise à la compétition CAESAR.







# Contents

<b>Remerciements</b>	<b>i</b>
<b>Résumé en français</b>	<b>v</b>
Chiffrement de base de données . . . . .	vi
Travaux sur les algorithmes de recherche sur bases de données chiffrées . . . . .	vii
Autres travaux . . . . .	x
<hr/>	
<b>1 Introduction</b>	<b>3</b>
1.1 The Need for Encrypted Databases . . . . .	4
1.2 History of Searchable Encryption . . . . .	5
1.3 Contributions of this Thesis . . . . .	12
<b>2 Notation, Definitions and Preliminaries</b>	<b>23</b>
2.1 Mathematical Notations . . . . .	24
2.2 Cryptographic Preliminaries . . . . .	26
2.3 Cryptographic Primitives . . . . .	29
<b>3 Basics of Searchable Encryption</b>	<b>39</b>
3.1 Definitions . . . . .	40
3.2 Leakage in Searchable Encryption . . . . .	48
3.3 The Locality of Searchable Encryption . . . . .	56
3.4 Leakage Abuse Attacks . . . . .	59
<b>4 Forward Privacy</b>	<b>65</b>
4.1 File Injection Attacks . . . . .	66
4.2 Definition of Forward Privacy . . . . .	67
4.3 Constraints Induced by the Forward Privacy . . . . .	68
4.4 Building a Forward Private SSE Scheme from Static Schemes . . . . .	71
4.5 $\Sigma\phi\phi\phi$ : Simple Optimal Forward Secure Searchable Encryption . . . . .	72
4.6 Diana: Forward-Secure SSE with Very Low Overhead . . . . .	82
4.7 Implementation and Evaluation of $\Sigma\phi\phi\phi$ and Diana . . . . .	88
<b>5 Backward Privacy</b>	<b>97</b>
5.1 Definition of Backward Privacy . . . . .	98
5.2 A Generic Two Round-trips Backward-Private Scheme . . . . .	100

5.3	Diana <sub>del</sub> : Backward Privacy from Range-Constrained and Puncturable PRFs . . . . .	103
5.4	Janus: Weak Backward Privacy from Puncturable Encryption . . . . .	104
<b>6</b>	<b>Verifiable Searchable Encryption</b>	<b>119</b>
6.1	A Lower Bound on Verifiable Searchable Encryption . . . . .	120
6.2	Tools for Constructing Verifiable Dynamic SSE schemes . . . . .	122
6.3	A Generic and Optimal Construction . . . . .	127
6.4	Verifying $\Sigma\phi\phi\phi$ , Diana, and Janus . . . . .	129
6.5	Verifying SPS . . . . .	130
<b>7</b>	<b>Leakage Abuse Attacks and How to Thwart Them</b>	<b>151</b>
7.1	Leakage Abuse Attacks and their Origin . . . . .	152
7.2	Fixing the Security Definition . . . . .	153
7.3	Keywords Clustering . . . . .	160
7.4	Application to Database Padding with Best Possible Security . . . . .	163
7.5	Experiments . . . . .	170
<b>8</b>	<b>Conclusion</b>	<b>177</b>
8.1	Summary of the Results . . . . .	177
8.2	Open Problems . . . . .	178
	<b>Bibliography</b>	<b>181</b>
	<b>List of Figures</b>	<b>193</b>
	<b>List of Tables</b>	<b>195</b>
	<b>List of Algorithms</b>	<b>197</b>



Call me Ishmael

*Moby-Dick* – HERMAN MELVILLE

# Introduction 1



SEARCHABLE ENCRYPTION aims at making efficient a seemingly easy task: outsourcing the storage of a database to an untrusted server, while keeping search features. With the development of Cloud storage services, for both private individuals and businesses, efficiency of searchable encryption is crucial: inefficient constructions would not be deployed at a large scale because they would not be usable. The key problem with searchable encryption is that any construction achieving ‘perfect security’ induces a computational or a communication overhead that is unacceptable for the cloud providers or for the cloud users – at least with current techniques and by today’s standards.

In this chapter, we will more precisely describe the motivations behind searchable encryption and different solutions which have been proposed in the past.

## Contents

---

<b>1.1</b>	<b>The Need for Encrypted Databases</b>	<b>4</b>
<b>1.2</b>	<b>History of Searchable Encryption</b>	<b>5</b>
1.2.1	Encrypted Databases from Generic Tools	5
1.2.2	Single-keyword Searchable Encryption	8
1.2.3	Encrypted Databases Supporting Complex Queries	11
<b>1.3</b>	<b>Contributions of this Thesis</b>	<b>12</b>
1.3.1	$\Sigma\phi\phi\sigma$ : Forward Secure Searchable Encryption	12
1.3.2	Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives	13
1.3.3	Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security	13
1.3.4	Thwarting Leakage Abuse Attacks against Searchable Encryption – A Formal Approach and Applications to Database Padding	13
1.3.5	Original Contributions	14

---

## 1.1 The Need for Encrypted Databases

For the last decades, a considerable amount of data has been progressively outsourced from the data owners to external service providers, starting with emails, and extending to banking information, consumer information, or private pictures. More and more businesses decide to use collaborative platforms in their day-to-day work. Many companies also do not want to manage local infrastructures to store the ever growing amount of information they deal with. Outsourced infrastructure providers, such as Google, Amazon, Apple, Microsoft, IBM, Dropbox, and many others, offer a wide range of services — covering email, file hosting, collaborative documents edition, backups, and more — generically called ‘the Cloud’. These services are very appealing to private companies, to individuals, and even to the public sector, as they allow their users to ‘focus on the data, rather than on the infrastructure’, at a very competitive pricing<sup>1</sup>, and with good uptime and safety guarantees. Prospective reports state that, by 2020, nearly 80 % of the Small & Medium Businesses (SMBs) in the United States of America will use Cloud services, while ‘only’ 43 % did in 2015, that, in 2018, the European SMB Cloud service market will reach €30.1B [Col15], and that Cloud computing is expected to grow by 19 % by 2020 [GM16].

Note that Cloud services are not limited to data storage or data management, and cover the outsourcing of computations: a user who cannot afford a large computer can buy some computation time on Amazon Elastic Compute Cloud (EC2) or Google Compute Engine, and even very repetitive micro-tasks involving human intelligence can be crowdsourced using Amazon Mechanical Turk.

During the period, data protection has become a very important question for many companies dealing with massive amounts of sensitive data. The first reason for that is the development of dedicated legislation obliging companies to secure their data, in particular personal data. Example of such regulations are the Title II of the Health Insurance Portability and Accountability Act (HIPAA) of 1996 [USA96] in the healthcare sector or the Sarbanes–Oxley (SOX) Act [USA02] in the financial sector in the United States of America, and, in the European Union, the General Data Protection Regulation (GDPR) [EU16] — replacing the Data Protection Directive [EU95]. These laws intend to prevent any disclosure or access, either accidental, unlawful or unauthorized, to sensitive data. In most cases, this implies the use of encryption for data at rest, to ensure their confidentiality, but also the use of cryptographic techniques — user authentication, data at rest integrity — to prevent (undetected) data tampering. Also, with these regulations, outsourcing the storage of financial data or the processing of healthcare data to the Cloud without relying on cryptographic access control mechanisms is simply illegal, and many Cloud service providers are hence denied the access to huge market shares.

New regulation is not the only origin of the development of encrypted databases. After the 2013 revelations by Snowden [Gre14] of the electronic interception program of the United States’ National Security Agency (NSA) — helped by and in association with other, non American, intelligence agencies such as the GCHQ,<sup>2</sup> the ASD,<sup>3</sup> or the BND<sup>4</sup> — many Cloud users, and more generally internet users, both companies and individuals, got convinced of the necessity of encrypting their communications and their data in order to protect their privacy and/or their business. In particular, service providers are no longer trusted, as they can get subpoenaed to reveal encryption keys: the provider becomes the ‘adversary’, cryptographically speaking.

---

<sup>1</sup>At the time these lines are written (2017).

<sup>2</sup>Government Communications Headquarters (United Kingdom).

<sup>3</sup>Australian’s Signals Directorate (Australia).

<sup>4</sup>Bundesnachrichtendienst (Germany).



In this setting, only protecting the communications between the user and the server, say using TLS<sup>5</sup> to establish a secure channel, is not enough as the server will still see the data the client sent in cleartext. Indeed, modern instant messaging services use end-to-end-encrypted (E2EE) protocols which ensure that only the communicating parties can read the messages they exchange. The Signal protocol [MP16; CCD+17] is an example of such protocols, and has seen its use massively generalized since its adoption by the WhatsApp instant messaging application [KA16].

For databases, we similarly want a system preventing the server to infer information, either about the outsourced dataset, or about the accesses of the client, as both can be very sensitive. It is very important that the security of the scheme can be proven using standard cryptographic assumptions, as its users must be confident in its claimed security. And, in addition to being secure, we also want this system to be usable in practice. This means that it has to be easy to use (*i.e.* easy to deploy and supporting a set of features similar to the one of its insecure counterpart), that its performance must be competitive with an unencrypted database, and that it has to be scalable and support large datasets (up to terabytes of data). Otherwise, the secure system will never be chosen in preference to non-encrypted systems: Signal and WhatsApp are far more successful than PGP because these mobile applications are trivial to use, the opposite of PGP-encrypted emails.

We will see in this thesis that achieving both security and efficiency for simple functionalities is already delicate, and sometimes just impossible. Namely, we will focus on encrypted databases supporting single-keyword search — a.k.a. *searchable encryption* schemes.

## 1.2 History of Searchable Encryption

In the following paragraphs, we give an overview of various proposed constructions for encrypted databases, ranging from “perfectly” secure but inherently inefficient constructions, to more *ad hoc* schemes supporting a wide type of queries.

### 1.2.1 Encrypted Databases from Generic Tools

As for many secure functionalities, encrypted databases can be constructed from generic cryptographic tools. These tools are very powerful and solve our problem perfectly in terms of security. Yet, we will see that, in some sense, they are *too* powerful: perfect security comes at an unaffordable cost, either in terms of computation or in terms of communication.

**Private information retrieval (PIR).** The first tool that one can think of when looking at encrypted databases is private information retrieval. A PIR protocol allows a user to retrieve an element of a database from a server, without the server learning which element was accessed. Note that, in the PIR setting, the database is public (and hence known by the server).

PIR has been introduced by Chor *et al.* in [CGKS95], and many different flavors have been studied since then.

Firstly, one can consider information theoretic PIR: the security of these protocols does not rely on any hardness assumption — they achieve perfect security. However, in this case, it is necessary to distribute the database to several non-colluding servers if the user does not want to download the entire database at every query [CGKS98, Section 5]. In practice, this would require to rely on different service providers, and still, the queries would not be kept secret if all the providers are simultaneously attacked or subpoenaed.

---

<sup>5</sup>Transport Layer Security. See [KPW13] for more information on the security offered by TLS.

Another interesting candidate is computationally-Private Information Retrieval [KO97] (cPIR). In this case, the security relies on cryptographic assumptions, allowing for single-database schemes whose bandwidth overhead is sublinear in the size of the data set. However these schemes trade a low bandwidth for a high computational complexity: all the existing schemes perform  $\Omega(n)$  public key operations in order to answer to a query on a database of size  $n$  – with the exception of Lipmaa’s construction [Lip10] which only needs  $\mathcal{O}(n/\log n)$  operations. This prevents PIR from being able to scale to very large data sets, even when the implementation is very efficient, like in the scheme of Aguilar *et al.* [ABFK14]: with this implementation, the query latency (without accounting for the network) is of the order of tens of seconds for a database with 100 000 documents.

Finally, symmetric PIR [GIKM98] (also called Oblivious Transfer [NP99]), where the sender of the query cannot learn more information than what he asked for, might be interesting for multi-user encrypted databases with fine-grained access control. However, symmetric PIR suffers from the same inefficiencies as the non-symmetric constructions, and cannot be reasonably used.

We refer the curious reader to the surveys by Gasarch [Gas04] and by Ostrovsky and Skeith [OS07] for more information about PIR.

**Fully homomorphic encryption (FHE).** The idea of homomorphic encryption is that given ciphertexts  $c_1$  and  $c_2$  encrypting respectively  $m_1$  and  $m_2$ , anyone can compute the encryption of  $f(m_1, m_2)$  for a given function  $f$ , without using the secret key. El Gamal [ElG84] and (textbook) RSA [RSA78] encryption schemes are examples of *multiplicatively homomorphic* schemes ( $f$  is the multiplication modulo a fixed integer  $N$ ), and Goldwasser-Micali [GM84] and Paillier [Pai99] cryptosystems are *additively homomorphic* ( $f$  is the addition modulo 2 and  $N$ , respectively). When any computable function  $f$  can be evaluated homomorphically by an encryption scheme, we say that it is *fully homomorphic*.

Homomorphic encryption was introduced by Rivest, Adleman and Dertouzos as ‘privacy homomorphism’ [RAD78]. The first *fully* homomorphic encryption scheme has been designed by Gentry [Gen09], and since then, a lot of work on this topic arose [DGHV10; BGV12; GSW13; CGGI16], making FHE schemes better understood and more practical.

Fully homomorphic encryption has numerous applications, especially for Cloud services, starting with the outsourcing of computation. Database outsourcing was actually the primary motivation for homomorphic encryption in the introductory paper by Rivest, Adleman and Dertouzos.

With FHE, the database would be encrypted by the client, and given to the server. Then the client would encrypt its query (or at least the query terms), and the server will evaluate the query on the encrypted database to compute the result, that will be sent back to the client for decryption. Yet this approach cannot be asymptotically efficient: to be secure, the FHE evaluation procedure has to depend on every bit of the database. Otherwise, the adversary would be able to learn some information about the dataset or about the query, and possibly to break the security of the encryption scheme, by observing the running time of the query. Accordingly, the computational complexity of the query is linear in the size of the dataset, while, on non encrypted systems, the running time of the query is linear in the number of results (every result has to be sent back to the client) and in the size of the query (all the query has to be processed).

To overcome this limitation, Boneh *et al.* [BGH+13] proposed an homomorphic-encryption-based scheme, in which the database server is split in two non-colluding parties, the ‘server’ and the ‘proxy’, supporting complex queries. Yet, similarly to symmetric PIR, this construction is meant to protect the privacy of the queries, and the confidentiality of the database for non-returned results, and does not really fit our needs for secure database outsourcing.

**Multiparty computation (MPC).** The goal of secure multiparty computation is to securely compute a function whose inputs are distributed among several parties. In particular, the parties want to keep their input private. Also, one might want to ensure the correctness of the result when the parties are malicious and do not follow the protocol.

Two-party computation (2PC) has been introduced by Yao [Yao82; Yao86], with the famous Millionaires problem — namely securely comparing two integers — and later extended to the multiparty case by Goldreich, Micali and Wigderson [GMW87]. In particular, any function (represented as a circuit) can be securely evaluated using MPC. To do this, Goldreich *et al.* use two very important and interesting cryptographic tools: oblivious transfer (OT) — already mentioned in a previous paragraph — and garbled circuits.

The idea of garbled circuit, originating from lectures by Yao, is to ‘encrypt’ the truth table of each gate of the circuit computing the function. Then, a party can use these encoded gates to evaluate the circuit using some garbled inputs provided by the garbling party. Many improvements have been developed over the past years to improve the efficiency of garbled circuits [BMR90; KS08; FGK17], making them practical for real-world applications [GLNP15; ABF+17].

Yet, garbled circuits, and MPC in general, are not really well suited to encrypted search applications. Namely, in searchable encryption, only one party has the secret inputs and the other is here to help it with the computations, while, in MPC applications, there are multiple parties, each with its own private input. Also, garbled circuits can only be used once<sup>6</sup> and directly using garbled circuits to answer search queries would require a mechanism to re-garble the database, in a private way, at every query. Finally, as for FHE, the circuit representation of the query function would induce a large overhead, linear in the size of the database.

**Oblivious RAM (ORAM) and oblivious data structures.** Another way to construct searchable encryption is to implement a regular search engine on top of (secure) outsourced storage: every storage access (*i.e.* every read/write operation) needed when the client runs the search engine is processed to ensure it leaks no information.

Yet this approach is not completely straightforward: the way the search engine processes queries might depend on the query itself and hence could reveal some crucial, side-channel information about the query. Also, the sequence of accessed locations in the outsourced memory should not reveal any information: the access pattern must be concealed.

The goal of Oblivious RAM is exactly to hide this data access pattern from the server storing this data as a sequence of blocks. Said otherwise, from the server’s perspective, the access pattern of two sequences of read/write operations of the same length are indistinguishable. ORAM has been introduced by Goldreich and Ostrovsky in [GO96], in which the authors proposed a scheme based on shuffling and re-encrypting data blocks. Since then, the research has been very active in this field, and many schemes were developed, which we could roughly classify in two families: hierarchical/square root ORAM — similar to first constructions by Goldreich and Ostrovsky [GO96] — and tree-based ORAM — with notably the Path ORAM construction by Stefanov *et al.* [SDS+13a].

Many variations of strict ORAM have been developed, such as multiple-server ORAM [SS13; LO13a], or use of some server computational power [WS12; DDF+16; GMP16]. Note that assuming some server computing power is very appealing, first because, in our encrypted database case, we do suppose that the server can do some computation — even unencrypted datasets require some work from the server — and then because this allows to ‘break’ the ORAM lower bound stated in the

---

<sup>6</sup>There has been some work on reusable garbled circuits (*cf.* [GKP+13]) but these constructions are absolutely not practical.

seminal work of Goldreich and Ostrovsky in [GO96]: this paper shows that an ORAM of  $N$  blocks with constant client storage incurs a client-server bandwidth overhead of  $\mathcal{O}(\log N)$ , in the case of server doing no computation (think of a disk or a RAM module). When the server is allowed to help the client, this lower bound does not apply anymore to the bandwidth between the client and the server, but between the server CPU and its local storage, which is much less of a problem in Cloud scenarios, but still a limitation.

Yet, as explained above, using secure memory accesses does not prevent side-channel leakage from the length of access pattern. The algorithms used to answer the queries must be designed carefully to tackle this issue. Also, by carefully studying how the ORAM protocol and the query algorithms interact, we can also improve on the efficiency of the combined construction.

One way to proceed is to use Oblivious Data Structures (ODS) [WNL+14]: in this work, Wang *et al.* use the Path ORAM construction in a very particular way, so as to limit the client-server bandwidth overhead. Similarly, a variant of Path-ORAM combined with server-evaluated garbled circuits – TWORAM – has been used to construct a searchable encryption scheme [GMP16]. Finally, Garbled RAM [LO13b] is a technique based on ORAM to evaluate RAM programs on an untrusted third party (*i.e.* with a constant number of client-server interactions), which can be applied to searchable encryption but at a very high cost: the Garbled RAM compiler of Lu and Ostrovsky incurs a computational overhead of  $\text{poly}(\lambda, \log N)$ , where  $N$  is the size of the inputs. Finally, ORAM has been adapted to improve the practical performance of secure computations (by reducing the ORAM circuit size) [WHC+14], so as to allow fast access to randomly accessible memory within a garbled circuit.

Note that all the ORAM-based constructions of searchable encryption will leak the number of memory accesses they perform. Hence, they often leak the number of results of a search query, which is not the case with FHE-based or PIR-based schemes. Yet, they will be a lot more efficient as their running time can depend on the number of results only, while all the FHE/PIR-based constructions necessarily run in time linear in the size of the database.

**Leakage is necessary.** Despite being very powerful, all these generic techniques suffer from inherent inefficiencies: FHE and MPC are based on circuits, and ORAM-based constructions will be subject to a lower bound on their efficiency.

So, how to make searchable encryption efficient, both in theory and in practice? How can these lower bounds be overcome? The answer is that we have to trade security for efficiency: some information must leak to the server when he runs a query. Note that we already mentioned this phenomenon: ORAM-based systems can be more efficient than FHE-based constructions because they can run in time depending on the number of results (as a RAM program), while, as FHE hides everything about the outcome of the computation, its running time of an homomorphic evaluation is always the running time of the worst case.

In Section 3.1, we will see how we can formally define the leakage of searchable encryption schemes, and in Section 3.2, we will describe the leakage of some commonly encountered schemes. In particular, we will show that some basic leakages are necessary for any searchable encryption scheme whose asymptotic performances are similar to the ones of its unencrypted counterpart, even for schemes supporting very basic queries (single-keyword search).

### 1.2.2 Single-keyword Searchable Encryption

The first searchable encryption construction dates from 2000 and a paper of Song, Wagner and Perrig [SWP00]. The authors construct a kind of stream cipher to encrypt documents, so that the

ciphertext can be partially decrypted using keyword-derived tokens. To perform a search on a keyword, the client sends the corresponding token and the server tries to decrypt the segments of the ciphertext: if the decryption succeeds, it is a match. Unfortunately, used as this, this scheme is quite inefficient because a search requires work linear in the size of the database, and hence is not scalable.

However, the paper [SWP00] is very important: it already sketches and/or underlines many interesting points about searchable encryption. Firstly, it provides a provably secure construction, in the sense that their scheme is a secure encryption scheme for the database: it is proven that the document encryption function is a secure pseudo-random generator, and hence that the encrypted documents leak no information by themselves. Then, it considers using an index to speed up the search without compromising the security, and already notes the issues with indexes that subsequent works tried to tackle: complexity of the update, necessity to pad the encrypted database to hide the frequency of keyword. Also, the authors mention what will later be called leakage-abuse attacks, *i.e.* the fact that the adversary might learn a lot of information about the database, beyond the ones directly leaked by the queries, and describe ways to mitigate these attacks, and to counter misbehaving adversaries trying to return incorrect query answers.

Goh [Goh03], and Chang and Mitzenmacher [CM05] specifically studied encrypted indexes, and focused on security definitions (*cf.* Section 3.1). Goh's scheme consists in storing a Bloom filter per document — a filter which can be tested to check if a keyword belongs to this particular document. The search complexity of the scheme is then linear in the number of documents, and, because the size of the Bloom filters is chosen so that they can contain every keyword, the size of the encrypted database is  $D \cdot K$  where  $D$  is the number of documents and  $K$  the number of keywords, which can be huge compared to the optimal size when the dataset is sparse (the documents contain only a small subset of all the keywords).

Chang and Mitzenmacher's scheme encrypts a bit matrix, in which the rows represent the documents and the columns the keywords. A bit is set to 1 if the keyword of the corresponding column appears in the document corresponding to the row. The matrix is then encrypted column per column, so that, during a query, the client can give the decryption key corresponding to the searched keyword. The server simply has to decrypt this column to find the matching documents. This time, the search time of the scheme is linear in the total number of documents in the database (which is sub-optimal) and, again, the size of the encrypted database is  $D \cdot K$ .

Modern security definitions were introduced by Curtmola *et al.* in [CGKO06]. They were the first to formally introduce the notion of leakage in these definitions. Their security definition is also the first one to consider the security of the encrypted database and the security of the search tokens together, not separately. In particular, they showed that a two-part definition, separating these two notions, is most likely to fail. All this allowed them to design an asymptotically efficient index-based scheme. This scheme, which is a kind of encrypted multi-map implemented using an implicit linked-list, inspired or served as a basis of many of the later single (and multiple) keyword SSE constructions (*e.g.* [CK10; KPR12; KO12; CJJ+13; PBP16; Bos16; KM17] and more).

An important challenge for the searchable encryption community has been the development of dynamic schemes, *i.e.* schemes supporting modifications of the encrypted database *via* an update mechanism. Until the work of Kamara, Papamanthou and Roeder [KPR12], there was no efficient dynamic searchable encryption scheme. By 'efficient', we mean a scheme whose search complexity is sublinear in the number of documents, whose encrypted database size is linear in the size of the plain dataset, and whose update complexity is linear in the number of updated document/keyword pairs. The scheme of Kamara *et al.* is based on the encrypted-linked-list-based multimap of [CGKO06]

and performs insertions by (logically) enqueueing each new entry in the right list. Efficient deletions are supported by encoding a dual representation of the index: also using linked lists, the client can easily tell the server which tokens of the encrypted database correspond to a given document, and hence tell him to remove them when needed.

Unfortunately, as we will see in Chapter 4, dynamism allows for new means of attack, and regular dynamic index-based schemes, such as the ones in [KPR12; CJJ+14], are subject to devastating adaptive attacks (*cf.* [ZKP16]). To thwart these attacks, forward-private schemes have been constructed [SPS14; Bos16], *i.e.* schemes whose update protocol is oblivious and reveals no information.

Also, for most dynamic schemes, a deletion only logically removes the entries, but the server is still able to decrypt these deleted entries — and hence extract some supposedly erased information — for any subsequent search request matching these. However, this problem can be fixed using backward private constructions (*cf.* [BMO17]), as we will see in Chapter 5.

Many of the previously mentioned schemes are asymptotically optimal: the complexity of the search protocol is linear in the number of results, the complexity of the update protocol is linear in the number of updates, the storage overhead of encrypted database is constant. Unfortunately, the performance of these schemes is still one order of magnitude behind that of an unencrypted databases. The main explanation for this is the non-locality of the storage accesses of secure searchable encryption schemes: these schemes perform many random memory accesses, while unencrypted databases are able to re-organize their data so that memory accesses are sequential, which are faster than random accesses by several orders of magnitude (*cf.* Section 3.3 for detailed discussions and results about locality in searchable encryption). A way to overcome this problem is to use a ‘legacy-compatible’ construction, *i.e.* a construction built on top of a regular, non-encrypted database. The encryption scheme can be seen as a proxy transforming keywords in documents into cryptographic tokens which are then placed in the database. Many commercial encrypted databases (*e.g.* CipherCloud [Cip] or skyhigh [sky]) are built in this manner. These legacy-compatible schemes use deterministic encryption, and hence are subject to attacks based on frequency-analysis [NKW15], compromising the secrecy of the encrypted dataset. In practice, these constructions bring very little security. Also note that it is possible to construct encrypted databases supporting more complex queries than single-keyword search as legacy-compatible schemes — *cf.* next section.

Almost all of the constructions mentioned above are shown secure in a model where the adversary has to follow the prescribed protocols. Yet, in real-world systems, it is also important — if not essential — to ensure that the adversary does not cheat, and does not derive from the protocols, *e.g.* by returning an incomplete set of results. There are many examples, in the applied cryptography literature, where failing to properly achieve integrity leads to breaks in the confidentiality of private data (*cf.* [Vau02; BL16]).

For searchable encryption, this problem was first studied by Kurosawa and Ohtaki [KO12] in the setting of static databases. They construct a scheme that is secure against malicious adversaries in the Universal Composability (UC) model [Can01], a very strong and very powerful formalization for cryptographic functionalities. Their idea is to use a message authentication code over the result set for each keyword to ensure that the adversary has not tampered it. This can be done because the result set for each keyword is pre-computed during the indexing phase necessary to build the index-based encrypted database. Kurosawa and Ohtaki later extended their construction to dynamic databases in [KO13].

In Chapter 6, we will study in more details the problem of searchable encryption secure against malicious adversaries, *a.k.a.* *verifiable searchable encryption*, and in particular see that there is a

tight lower bound on the efficiency of these constructions (*cf.* [BFP16]).

### 1.2.3 Encrypted Databases Supporting Complex Queries

Although supporting single-keyword searches is a first step, it is not sufficient for all database applications. To be usable, the query language of the encrypted database must extend to more complex queries. Again, such features can be implemented using generic techniques, with no or little leakage, but at huge computational or communication cost.

Solutions built on top of index-based searchable encryption have been developed to support boolean queries, *i.e.* queries consisting of conjunctions, disjunctions, or negations of keywords (*e.g.* “tiger AND (bengal OR siberian)” to retrieve only the pictures of Bengal or Siberian tigers in a picture database). The first contributions of this kind have been concurrently authored by Cash *et al.* [CJJ+13] – OXT – and by Pappas *et al.* [PKV+14]<sup>7</sup> – Blind-Seer.

The OXT construction is particularly targeted towards conjunctive queries. It combines a single-keyword scheme, used for the first term of the conjunction, an oblivious pseudorandom function (OPRF), to securely compute tokens from the indices of the documents matching the first query term and the other terms, and finally, a set-membership data structure (*e.g.* a Bloom filter), used to test if a document/keyword pair, represented by an OPRF-computed token, belongs to the dataset.

Blind Seer is based on a completely different design: the inverted index is encoded using a tree of Bloom filters, inspired from a construction of Goh [Goh03]. Each leaf corresponds to a document, and the corresponding Bloom filter contains all the records the document has. The rest of the tree is based on the invariant that its Bloom filter contains all the records contained by its children. In this tree based construction, a search corresponds to a tree traversal based on Bloom filter testing. In this data structure, it is also quite easy to answer boolean queries by adapting the tree traversal predicate. The security of the construction comes from the encryption of the Bloom filters (using pseudo-randomly generated one-time-pad) and the evaluation of the tree-traversal predicate with Garbled circuits.

More recently, Kamara and Moataz [KM17] presented a scheme supporting monotone boolean queries, and in particular disjunctive queries, with an optimal search complexity.

It is interesting to note that, from boolean queries, researchers have been able to build schemes supporting range [PKV+14], wildcards, substrings or phrases [FJK+15] queries. Also, some schemes address the problem of fuzzy encrypted search, *i.e.* search on approximate keywords, tolerating typos and enabling biometrics applications – *cf.* [BC15].

Yet, these constructions are still far from SQL-like databases, either in terms of queries’ expressiveness or in terms of performance. At the same time, new legacy-compatible constructions appeared to bridge this gap. The most famous one is probably CryptDB [PRZB11], supporting aggregate, range and equality queries. To achieve these features, CryptDB encrypts the columns of the database using homomorphic encryption, deterministic encryption, or order-preserving encryption.

Order-preserving encryption (OPE) is a (possibly deterministic) encryption scheme whose encryption function preserves the order of the plaintexts. It is easy to see that OPE can be used as a component of a legacy-compatible encrypted database supporting range-queries. OPE was formally introduced by Boldyreva *et al.* [BCLO09], and, in association with its relaxation order-revealing encryption (ORE) – in ORE the numerical order of plaintext is not preserved, but ciphertext can be efficiently compared using a public algorithm – has been an active topic [PLZ13; Ker15; CLWW16; RACY16].

<sup>7</sup>Both of these constructions are results of a DARPA Project.

However, because deterministic encryption and OPE/ORE are far more leaky than regular, semantically secure encryption, these legacy-compatible constructions are often easy to break. Recently, attacks have been presented breaking CryptDB [NKW15], and the OPE scheme of ARX [LMP17] (CryptDB’s successor [PBP16]). Some non-legacy-compatible constructions have been designed (e.g. [KM16]), offering an security improvement over the previous schemes. Unfortunately, the leakage description of these is quite involved and has not really been (crypt)analyzed. The curious reader should consider the excellent Systematization of Knowledge paper on encrypted database by Fuller *et al.* [FVY+17].

Another interesting feature of encrypted search, especially when the number of results of a query is large, is the ranking of results. Once again, the client could do the ranking by himself, based on some relevance information stored in the database. But, when the result set is huge, and the client only wants the first 10 results (think about a web search engine), this is not reasonable. To solve this problem, Baldimtsi and Ohrimenko [BO15], and Meng *et al.* [MZK15] gave constructions, respectively sorting the results and returning the top-k results of a boolean search query. These schemes are built in a model assuming the existence of two non-colluding cloud providers. The first one stores the encrypted database and executes the query, while the second ranks or selects the top answers using some client-issued private keys.

Besides searchable encryption, some works focused on generalizations of encrypted databases to other representations of data. Structured encryption was introduced in [CK10] as a generalization of searchable encryption to any data structure. In this paper, Chase and Kamara gave constructions for neighbor and adjacency queries on graph-structured data, as well as lookup queries on matrix-structured data. Similarly, encrypted data structures able to answer approximate shortest distance queries in a graph have been developed by Meng *et al.* [MKNK15].

## 1.3 Contributions of this Thesis

This thesis studies single-keyword searchable encryption. It brings in at once new theoretical results, new security considerations, new constructions, and new practical analysis of searchable encryption. These results mainly come from four papers, that we will quickly summarize, and a few others are original.

### 1.3.1 $\Sigma\phi\phi\phi$ : Forward Secure Searchable Encryption [Bos16]

Recent work [ZKP16] showed that dynamic schemes — in which the data is efficiently updatable — leaking some information on updated keywords are subject to devastating adaptive attacks breaking the privacy of the queries. The only way to thwart this attack is to design *forward private* schemes whose update procedure does not leak if a newly inserted element matches previous search queries.

The [Bos16] paper formalized the notion of forward privacy and proposed  $\Sigma\phi\phi\phi$  as a forward private SSE scheme with performance similar to existing less secure schemes, and that is conceptually simpler (and also more efficient) than previous forward private constructions. In particular, it only relies on trapdoor permutations and does not use an ORAM-like construction. We also explain why  $\Sigma\phi\phi\phi$  is an optimal point of the security/performance tradeoff for dynamic SSE (supporting only insertions, but not deletions). Finally, an implementation and evaluation results demonstrated its practical efficiency.

The material of this paper appears in Chapter 4.



### 1.3.2 Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives [BMO17]

With the previous paper, we saw why it is desirable that updates in searchable encryption schemes do not reveal any information *a priori* about the modifications they carry out. It would also be nice (and better) if deleted results remained inaccessible to the server *a posteriori*. This second property, called *backward privacy*, had been overlooked in previous work.

With Brice Minaud and Olga Ohrimenko, we studied for the first time the notion of backward privacy for searchable encryption. After giving formal definitions for different flavors of backward privacy, we presented several schemes achieving both forward and backward privacy, with various efficiency tradeoffs.

Our constructions crucially rely on primitives such as constrained pseudo-random functions and puncturable encryption schemes. Using these advanced cryptographic primitives allows for a fine-grained control of the power of the adversary, preventing her from evaluating functions on selected inputs, or decrypting specific ciphertexts. In turn, this high degree of control allows our SSE constructions to achieve the stronger forms of privacy outlined above. As an example, we present a framework to construct forward-private schemes from range-constrained pseudo-random functions, and the first backward-private searchable encryption scheme out of puncturable encryption. Finally, we provided experimental results for implementations of our schemes, and studied their practical efficiency.

The content of this paper is split between Chapter 4 (for the forward-private constructions) and Chapter 5 (for the backward privacy).

### 1.3.3 Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security [BFP16]

In this paper, we studied and designed the first efficient SSE schemes provably secure against *malicious* servers. First, we gave lower bounds on the complexity of such verifiable SSE schemes, and then constructed generic solutions matching these bounds using efficient verifiable data structures. Finally, we modified an existing scheme – SPS [SPS14], which also provides forward privacy of search queries – and made it provably secure against active adversaries, without increasing the computational complexity of the original scheme.

Chapter 6 takes this paper up.

### 1.3.4 Thwarting Leakage Abuse Attacks against Searchable Encryption - A Formal Approach and Applications to Database Padding [BF17]

As explained above, practically efficient searchable encryption leaks some information to the server. Many new attacks have recently been developed, targeting this leakage in order to break the confidentiality of the dataset or of the queries, through *leakage abuse attacks*.

These works helped to understand the importance of considering leakage when analyzing the security of searchable encryption schemes, but did not explain why these attacks were so powerful despite the existence of rigorous security definitions and proofs, or how they could be efficiently and provably mitigated.

Our paper addressed these questions by proposing an analysis of existing leakage abuse attacks and a way to capture them in new security definitions. These new definitions also helped us to devise a way to thwart these attacks and we applied it to the padding of datasets, in order to hide the

number of queries' results, and to provide provable security of some schemes with specific leakage profile against some common classes of leakage abuse attacks.

Finally, we gave experimental evidence that our countermeasures can be implemented efficiently, and easily applied to existing searchable encryption schemes.

This work is reproduced in Chapter 7.

### 1.3.5 Original Contributions

Two new results are also presented in this thesis. These are lower bounds on the efficiency of searchable encryption. We studied the search performance of the search algorithm, when the repetition of searches is kept secret for the first one, which appears in Chapter 3. The second result bounds the update efficiency of forward-private schemes, and is presented in Chapter 4.

### References of Contributions on Searchable Encryption

- [BF17] Raphael Bost and Pierre-Alain Fouque. *Thwarting Leakage Abuse Attacks against Searchable Encryption – A Formal Approach and Applications to Database Padding*. Cryptology ePrint Archive, Report 2017/1060. <http://eprint.iacr.org/2017/1060>. 2017 (cit. on pp. x, 13, 151).
- [BFP16] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. *Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security*. Cryptology ePrint Archive, Report 2016/062. <http://eprint.iacr.org/2016/062>. 2016 (cit. on pp. ix, 11, 13, 119).
- [BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. *Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives*. In: *ACM CCS 17*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1465–1482 (cit. on pp. viii, ix, 10, 13, 65, 97).
- [Bos16] Raphael Bost. *Σοφός: Forward Secure Searchable Encryption*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1143–1154 (cit. on pp. viii, 9, 10, 12, 56, 65).

### References of Other Contributions

- [BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. *Machine Learning Classification over Encrypted Data*. In: *NDSS 2015*. The Internet Society, Feb. 2015 (cit. on p. xi).
- [BS16] Raphael Bost and Olivier Sanders. *Trick or Tweak: On the (In)security of OTR's Tweaks*. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 333–353 (cit. on p. xi).

## References

- [ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. *Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier*. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 843–862 (cit. on p. 7).
- [ABFK14] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. *XPIR: Private Information Retrieval for Everyone*. Cryptology ePrint Archive, Report 2014/1025. <http://eprint.iacr.org/2014/1025>. 2014 (cit. on p. 6).
- [BC15] Alexandra Boldyreva and Nathan Chenette. *Efficient Fuzzy Search on Encrypted Data*. In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Heidelberg, Mar. 2015, pp. 613–633 (cit. on p. 11).
- [BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O’Neill. *Order-Preserving Symmetric Encryption*. In: *EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, Heidelberg, Apr. 2009, pp. 224–241 (cit. on p. 11).
- [BF17] Raphael Bost and Pierre-Alain Fouque. *Thwarting Leakage Abuse Attacks against Searchable Encryption – A Formal Approach and Applications to Database Padding*. Cryptology ePrint Archive, Report 2017/1060. <http://eprint.iacr.org/2017/1060>. 2017 (cit. on pp. x, 13, 151).
- [BFP16] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. *Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security*. Cryptology ePrint Archive, Report 2016/062. <http://eprint.iacr.org/2016/062>. 2016 (cit. on pp. ix, 11, 13, 119).
- [BGH+13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. *Private Database Queries Using Somewhat Homomorphic Encryption*. In: *ACNS 13*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, Heidelberg, June 2013, pp. 102–118 (cit. on p. 6).
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *(Leveled) fully homomorphic encryption without bootstrapping*. In: *ITCS 2012*. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325 (cit. on p. 6).
- [BL16] Karthikeyan Bhargavan and Gaëtan Leurent. *Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH*. In: *NDSS 2016*. The Internet Society, Feb. 2016 (cit. on p. 10).
- [BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. *Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives*. In: *ACM CCS 17*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1465–1482 (cit. on pp. viii, ix, 10, 13, 65, 97).
- [BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. *The Round Complexity of Secure Protocols (Extended Abstract)*. In: *22nd ACM STOC*. ACM Press, May 1990, pp. 503–513 (cit. on p. 7).
- [BO15] Foteini Baldimtsi and Olga Ohrimenko. *Sorting and Searching Behind the Curtain*. In: *FC 2015*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. LNCS. Springer, Heidelberg, Jan. 2015, pp. 127–146 (cit. on pp. 12, 179).

- [Bos16] Raphael Bost.  $\Sigma\phi\phi\sigma$ : *Forward Secure Searchable Encryption*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1143–1154 (cit. on pp. [viii](#), [9](#), [10](#), [12](#), [56](#), [65](#)).
- [BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. *Machine Learning Classification over Encrypted Data*. In: *NDSS 2015*. The Internet Society, Feb. 2015 (cit. on p. [xi](#)).
- [BS16] Raphael Bost and Olivier Sanders. *Trick or Tweak: On the (In)security of OTR’s Tweaks*. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 333–353 (cit. on p. [xi](#)).
- [Can01] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. In: *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145 (cit. on p. [10](#)).
- [CCD+17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. *A Formal Security Analysis of the Signal Messaging Protocol*. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 451–466. ISBN: 978-1-5090-1751-5. [Link](#). (Cit. on p. [5](#)).
- [CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds*. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 3–33 (cit. on p. [6](#)).
- [CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions*. In: *ACM CCS 06*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, Oct. 2006, pp. 79–88 (cit. on pp. [9](#), [42–45](#), [56](#), [72](#), [152](#)).
- [CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. *Private Information Retrieval*. In: *36th FOCS*. IEEE Computer Society Press, Oct. 1995, pp. 41–50 (cit. on p. [5](#)).
- [CGKS98] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. “Private Information Retrieval”. In: *Journal of the ACM* 45.6 (1998), pp. 965–982 (cit. on p. [5](#)).
- [Cip] CipherCloud. *Cloud Data Encryption*. [Link](#). (Cit. on p. [10](#)).
- [CJJ+13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 353–373 (cit. on pp. [9](#), [11](#), [43](#), [45](#), [58](#)).
- [CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. [10](#), [49](#), [56](#), [73](#), [83](#), [99](#), [129](#), [163](#), [170](#), [171](#)).
- [CK10] Melissa Chase and Seny Kamara. *Structured Encryption and Controlled Disclosure*. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 577–594 (cit. on pp. [9](#), [12](#), [43](#), [45](#), [57](#)).

- [CLWW16] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. *Practical Order-Revealing Encryption with Limited Leakage*. In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 474–493 (cit. on p. 11).
- [CM05] Yan-Cheng Chang and Michael Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*. In: *ACNS 05*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. LNCS. Springer, Heidelberg, June 2005, pp. 442–455 (cit. on pp. 9, 42).
- [Col15] Louis Columbus. *Roundup Of Small & Medium Business Cloud Computing Forecasts And Market Estimates, 2015*. May 2015. [Link](#). (Cit. on p. 4).
- [DDF+16] Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. *Onion ORAM: A Constant Bandwidth Blowup Oblivious RAM*. In: *TCC 2016-A, Part II*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9563. LNCS. Springer, Heidelberg, Jan. 2016, pp. 145–174 (cit. on pp. 7, 56).
- [DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43 (cit. on p. 6).
- [ElG84] Taher ElGamal. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 10–18 (cit. on p. 6).
- [EU16] *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. European Union, Apr. 2016. [Link](#). (Cit. on p. 4).
- [EU95] *DIRECTIVE 95/46/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data*. European Union, Oct. 1995. [Link](#). (Cit. on p. 4).
- [FGK17] Xiong Fan, Chaya Ganesh, and Vladimir Kolesnikov. *Hashing Garbled Circuits for Free*. In: *EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. LNCS. Springer, Heidelberg, May 2017, pp. 456–485 (cit. on p. 7).
- [FJK+15] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. *Rich Queries on Encrypted Data: Beyond Exact Matches*. In: *ESORICS 2015, Part II*. Ed. by Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl. Vol. 9327. LNCS. Springer, Heidelberg, Sept. 2015, pp. 123–145 (cit. on p. 11).
- [FVY+17] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. *SoK: Cryptographically Protected Database Search*. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 172–191 (cit. on p. 12).
- [Gas04] William Gasarch. “A survey on private information retrieval”. In: *The Bulletin of the EATCS* 82.72-107 (2004), p. 1 (cit. on p. 6).
- [Gen09] Craig Gentry. *Fully homomorphic encryption using ideal lattices*. In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178 (cit. on p. 6).

- [GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. *Protecting Data Privacy in Private Information Retrieval Schemes*. In: *30th ACM STOC*. ACM Press, May 1998, pp. 151–160 (cit. on p. 6).
- [GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. *Reusable garbled circuits and succinct functional encryption*. In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 555–564 (cit. on p. 7).
- [GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. *Fast Garbling of Circuits Under Standard Assumptions*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel: ACM Press, Oct. 2015, pp. 567–578 (cit. on p. 7).
- [GM16] John F. Gantz and Pam Miller. *The Salesforce Economy: Enabling 1.9 Million New Jobs and \$389 Billion in New Revenue Over the Next Five Years*. Sept. 2016. [Link](#). (Cit. on p. 4).
- [GM84] Shafi Goldwasser and Silvio Micali. “Probabilistic Encryption”. In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 270–299 (cit. on p. 6).
- [GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. *TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption*. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563–592 (cit. on pp. 7, 8, 45, 69, 73, 99, 101, 161, 163, 170).
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. *How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority*. In: *19th ACM STOC*. Ed. by Alfred Aho. ACM Press, May 1987, pp. 218–229 (cit. on p. 7).
- [GO96] Oded Goldreich and Rafail Ostrovsky. “Software protection and simulation on oblivious RAMs”. In: *Journal of the ACM* 43.3 (1996), pp. 431–473 (cit. on pp. 7, 8, 51, 52, 56, 82).
- [Goh03] Eu-Jin Goh. *Secure Indexes*. Cryptology ePrint Archive, Report 2003/216. <http://eprint.iacr.org/2003/216>. 2003 (cit. on pp. 9, 11, 42).
- [Gre14] Glenn Greenwald. *No place to hide: Edward Snowden, the NSA, and the US surveillance state*. Macmillan, 2014 (cit. on p. 4).
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 75–92 (cit. on p. 6).
- [KA16] Jan Koum and Brian Acton. *WhatsApp Blog: end-to-end encryption*. Apr. 2016. [Link](#). (Cit. on p. 5).
- [Ker15] Florian Kerschbaum. *Frequency-Hiding Order-Preserving Encryption*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel: ACM Press, Oct. 2015, pp. 656–667 (cit. on p. 11).
- [KM16] Seny Kamara and Tarik Moataz. *SQL on Structurally-Encrypted Databases*. Cryptology ePrint Archive, Report 2016/453. <http://eprint.iacr.org/2016/453>. 2016 (cit. on p. 12).

- [KM17] Seny Kamara and Tarik Moataz. *Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity*. In: *EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. LNCS. Springer, Heidelberg, May 2017, pp. 94–124 (cit. on pp. 9, 11).
- [KO12] Kaoru Kurosawa and Yasuhiro Ohtaki. *UC-Secure Searchable Symmetric Encryption*. In: *FC 2012*. Ed. by Angelos D. Keromytis. Vol. 7397. LNCS. Springer, Heidelberg, Feb. 2012, pp. 285–298 (cit. on pp. 9, 10, 47).
- [KO13] Kaoru Kurosawa and Yasuhiro Ohtaki. *How to Update Documents Verifiably in Searchable Symmetric Encryption*. In: *CANS 13*. Ed. by Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab. Vol. 8257. LNCS. Springer, Heidelberg, Nov. 2013, pp. 309–328 (cit. on pp. 10, 47).
- [KO97] Eyal Kushilevitz and Rafail Ostrovsky. *Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval*. In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 364–373 (cit. on pp. 6, 153).
- [KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. *Dynamic searchable symmetric encryption*. In: *ACM CCS 12*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. ACM Press, Oct. 2012, pp. 965–976 (cit. on pp. 9, 10).
- [KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. *On the Security of the TLS Protocol: A Systematic Analysis*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 429–448 (cit. on p. 5).
- [KS08] Vladimir Kolesnikov and Thomas Schneider. *Improved Garbled Circuit: Free XOR Gates and Applications*. In: *ICALP 2008, Part II*. Ed. by Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz. Vol. 5126. LNCS. Springer, Heidelberg, July 2008, pp. 486–498 (cit. on p. 7).
- [Lip10] Helger Lipmaa. *First CPIR Protocol with Data-Dependent Computation*. In: *ICISC 09*. Ed. by Donghoon Lee and Seokhie Hong. Vol. 5984. LNCS. Springer, Heidelberg, Dec. 2010, pp. 193–210 (cit. on p. 6).
- [LMP17] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. *Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage*. Cryptology ePrint Archive, Report 2017/701. <http://eprint.iacr.org/2017/701>. 2017 (cit. on p. 12).
- [LO13a] Steve Lu and Rafail Ostrovsky. *Distributed Oblivious RAM for Secure Two-Party Computation*. In: *TCC 2013*. Ed. by Amit Sahai. Vol. 7785. LNCS. Springer, Heidelberg, Mar. 2013, pp. 377–396 (cit. on p. 7).
- [LO13b] Steve Lu and Rafail Ostrovsky. *How to Garble RAM Programs*. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 719–734 (cit. on pp. 8, 56).
- [MKNK15] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. *GRECS: Graph Encryption for Approximate Shortest Distance Queries*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 504–517 (cit. on p. 12).
- [MP16] Moxie Marlinspike and Trevor Perrin. *The Double Ratchet Algorithm*. 2016. [Link](#). (Cit. on p. 5).

- [MZK15] Xianrui Meng, Haohan Zhu, and George Kollios. *Top-k Query Processing on Encrypted Databases with Strong Security Guarantees*. arXiv preprint arXiv:1510.05175. 2015 (cit. on p. 12).
- [NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. *Inference Attacks on Property-Preserving Encrypted Databases*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel: ACM Press, Oct. 2015, pp. 644–655 (cit. on pp. 10, 12, 59).
- [NP99] Moni Naor and Benny Pinkas. *Oblivious Transfer with Adaptive Queries*. In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 573–590 (cit. on p. 6).
- [OS07] Rafail Ostrovsky and William E. Skeith III. *A Survey of Single-Database Private Information Retrieval: Techniques and Applications (Invited Talk)*. In: *PKC 2007*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Vol. 4450. LNCS. Springer, Heidelberg, Apr. 2007, pp. 393–411 (cit. on p. 6).
- [Pai99] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238 (cit. on p. 6).
- [PBP16] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. *Arx: A Strongly Encrypted Database System*. Cryptology ePrint Archive, Report 2016/591. <http://eprint.iacr.org/2016/591>. 2016 (cit. on pp. 9, 12, 87, 99).
- [PKV+14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. *Blind Seer: A Scalable Private DBMS*. In: *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 359–374 (cit. on p. 11).
- [PLZ13] Raluca A. Popa, Frank H. Li, and Nikolai Zeldovich. *An Ideal-Security Protocol for Order-Preserving Encoding*. In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 463–477 (cit. on p. 11).
- [PRZB11] Raluca Ada Popa, Catherine Redfield, Nikolai Zeldovich, and Hari Balakrishnan. *CryptDB: protecting confidentiality with encrypted query processing*. In: *ACM SOSP 11*. ACM. 2011, pp. 85–100 (cit. on p. 11).
- [RACY16] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. *POPE: Partial Order Preserving Encoding*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1131–1142 (cit. on p. 11).
- [RAD78] Ronald Rivest, Leonard Adleman, and Michael Dertouzos. “On data banks and privacy homomorphisms”. In: *Foundations of secure computation* 4.11 (1978), pp. 169–180 (cit. on p. 6).
- [RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on pp. 6, 27).
- [SDS+13a] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. *Path ORAM: an extremely simple oblivious RAM protocol*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 299–310 (cit. on pp. 7, 56, 82).



- [sky] skyhigh. *Cloud Security and Enablement*. [Link](#). (Cit. on p. 10).
- [SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. [viii–x](#), [10](#), [13](#), [45](#), [59](#), [67](#), [71](#), [73](#), [81](#), [97–99](#), [115](#), [119](#), [122](#), [130–133](#), [141](#), [147](#), [163](#), [170](#)).
- [SS13] Emil Stefanov and Elaine Shi. *Multi-cloud oblivious storage*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 247–258 (cit. on p. [7](#)).
- [SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. *Practical Techniques for Searches on Encrypted Data*. In: *2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2000, pp. 44–55 (cit. on pp. [8](#), [9](#), [42](#)).
- [USA02] *Sarbanes-Oxley Act of 2002*. United States of America, July 2002. [Link](#). (Cit. on p. [4](#)).
- [USA96] *Health Insurance Portability and Accountability Act of 1996*. United States of America, Aug. 1996. [Link](#). (Cit. on p. [4](#)).
- [Vau02] Serge Vaudenay. *Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS...* In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, Apr. 2002, pp. 534–546 (cit. on p. [10](#)).
- [WHC+14] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. *SCORAM: Oblivious RAM for Secure Computation*. In: *ACM CCS 14*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM Press, Nov. 2014, pp. 191–202 (cit. on p. [8](#)).
- [WNL+14] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. *Oblivious Data Structures*. In: *ACM CCS 14*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM Press, Nov. 2014, pp. 215–226 (cit. on p. [8](#)).
- [WS12] Peter Williams and Radu Sion. *Single round access privacy on outsourced storage*. In: *ACM CCS 12*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. ACM Press, Oct. 2012, pp. 293–304 (cit. on p. [7](#)).
- [Yao82] Andrew Chi-Chih Yao. *Protocols for Secure Computations (Extended Abstract)*. In: *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164 (cit. on p. [7](#)).
- [Yao86] Andrew Chi-Chih Yao. *How to Generate and Exchange Secrets (Extended Abstract)*. In: *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 162–167 (cit. on p. [7](#)).
- [ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. *All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption*. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 2016, pp. 707–720 (cit. on pp. [10](#), [12](#), [60](#), [66](#), [76](#), [114](#), [152](#), [153](#), [156](#), [157](#)).

In the beginning, God created the heavens  
and the earth.

*Genesis*

# Notation, Definitions and Preliminaries

# 2



WE WILL NEED WELL FORMALIZED TOOLS throughout this thesis in order to have a formal reasoning. This small chapter defines the notational, mathematical and cryptographic building blocks necessary to the rest of the thesis, and that will be used for the subsequent definitions, theorems and proofs.

## Contents

---

<b>2.1</b>	<b>Mathematical Notations</b>	<b>24</b>
<b>2.2</b>	<b>Cryptographic Preliminaries</b>	<b>26</b>
2.2.1	Cryptographic Tools	26
2.2.2	Hardness Assumptions	27
<b>2.3</b>	<b>Cryptographic Primitives</b>	<b>29</b>
2.3.1	Pseudorandom Function	29
2.3.2	Constrained Pseudorandom Function	30
2.3.3	Pseudorandom Permutation	31
2.3.4	Trapdoor Permutation	31
2.3.5	Hash Function	32
2.3.6	Semantically Secure Encryption	34
2.3.7	Message Authentication Code	34
2.3.8	Authenticated Encryption with Associated Data	35

---

## 2.1 Mathematical Notations

**Sets and rings.** The set of integers is denoted  $\mathbb{Z}$ , the set of non-negative integers  $\mathbb{N}$ . If  $a$  and  $b$  are two integers such that  $a \leq b$ , we denote by  $\llbracket a, b \rrbracket$  the set  $\{x \in \mathbb{Z} \mid a \leq x \leq b\}$  of integers between  $a$  and  $b$  (both included). Also,  $\mathbb{Z}_n$  is the ring  $\mathbb{Z}/n\mathbb{Z}$  of integers modulo the integer  $n$ . For a prime integer  $p$ ,  $\mathbb{F}_p = \mathbb{Z}_p$  is the field with  $p$  elements.  $\varphi(\cdot)$  is the Euler totient function. For any set  $S$ ,  $\mathcal{P}(S)$  is the set of all finite subsets of  $S$ .

**Bilinear groups.** Some of the cryptographic primitives will use an additional structure on groups called a pairing (a.k.a. bilinear groups). Let  $\mathbb{G}_1, \mathbb{G}_2$  and  $\mathbb{G}_T$  be three cyclic groups of the same order  $N$  and with respective generators  $g_1, g_2$  and  $g_T$ . The quadruple  $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$  is called a *bilinear group* if  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  satisfies the following properties:

- For all  $(a, b) \in (\mathbb{Z}_N)^2$ ,  $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$  (bilinearity);
- The element  $e(g_1, g_2)$  generates  $\mathbb{G}_T$  (non-degeneracy);
- $e(\cdot, \cdot)$  is “efficiently” computable.

We explain the next paragraphs what “efficient” means. Without loss of generality, we may suppose that  $e(g_1, g_2) = g_T$ . We call bilinear groups with  $\mathbb{G}_1 = \mathbb{G}_2$  Type-1 (or symmetric) bilinear groups, and in this case, we let  $g = g_1 = g_2$ . If  $\mathbb{G}_1 \neq \mathbb{G}_2$ , and there is no efficiently computable isomorphism between  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , it is a Type-3 (asymmetric) bilinear group.

**Bit strings.** A bit is an element in  $\{0, 1\}$ ; a bit string  $s$  of length  $n$  is a vector of  $n$  bits. Let  $\varepsilon$  denote the *empty* bit string of size 0. The set of bit strings of size  $n$  is hence denoted  $\{0, 1\}^n$ , and the set of bit strings of finite length is  $\{0, 1\}^* = \cup_{n \geq 0} \{0, 1\}^n$ . The concatenation of two bit strings  $x$  and  $y$  is denoted  $x||y$ .

**Distributions and probabilities.** For a finite set  $S$ ,  $s \stackrel{\$}{\leftarrow} S$  means that  $s$  is sampled uniformly at random from  $S$ . The probability of an event  $E$  to occur is denoted  $\mathbb{P}[E]$ . Let  $X$  be a random variable. The expected value of  $X$  is  $\mathbb{E}[X]$ , and  $\text{Var}[X]$  is its variance. The *entropy* and *min-entropy* of a discrete random variable  $X$  taking value  $\{x_1, \dots, x_n\}$  are denoted  $H_1(X)$  and  $H_\infty(X)$  and are defined as

$$H_1(X) = - \sum_{i=1}^n \mathbb{P}[X = x_i] \cdot \log \mathbb{P}[X = x_i],$$

$$\text{and } H_\infty(X) = - \min_{1 \leq i \leq n} \{\log \mathbb{P}[X = x_i]\} = \max_{1 \leq i \leq n} \{-\log \mathbb{P}[X = x_i]\}$$

where  $\log$  is the base-2 logarithm. For two discrete random variables  $Y$  and  $Z$  over the set  $\{x_1, \dots, x_n\}$ , the *Kullback–Leibler divergence from  $Y$  to  $Z$* ,  $D_{\text{KL}}(Y||Z)$ , is defined as

$$D_{\text{KL}}(Y||Z) = \sum_{i=1}^n \mathbb{P}[Y = x_i] \log \frac{\mathbb{P}[Y = x_i]}{\mathbb{P}[Z = x_i]}.$$

Finally, we define the distance  $\Delta(\mathcal{D}_1, \mathcal{D}_2)$  between the two distributions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  over a set  $X$  as

$$\Delta(\mathcal{D}_1, \mathcal{D}_2) = \max_{x \in X} \{|\mathbb{P}[Y_1 = x] - \mathbb{P}[Y_2 = x]|\}$$

where  $Y_1$  (resp  $Y_2$ ) is a random variable following the distribution  $\mathcal{D}_1$  (resp.  $\mathcal{D}_2$ ).

**Asymptotics.** For asymptotics, we use the standard Landau notations  $\mathcal{O}(\cdot)$ ,  $\mathfrak{o}(\cdot)$ ,  $\Omega(\cdot)$ ,  $\omega(\cdot)$  and  $\Theta(\cdot)$ . We also use  $\tilde{\mathcal{O}}(\cdot)$  to hide poly-logarithmic factors in asymptotics:

$$f(n) = \tilde{\mathcal{O}}(g(n)) \Leftrightarrow \exists c \in \mathbb{N} \text{ such that } f(n) = \mathcal{O}(g(n) \log^c(n)).$$

In the following,  $\text{poly}(n)$  denotes an unspecified function  $f(n) = \mathcal{O}(n^c)$  for some fixed constant  $c$ , and  $\text{negl}(n)$  is a negligible function  $f$  such that  $f(n) = \mathfrak{o}(n^{-c})$  for any constant  $c > 0$ .

**Algorithms, Turing machines, and oracles.** Algorithms are programs for Turing machines. They may be probabilistic, in which case, they use a tape of the Turing machine filled with random bits (also called random coins). By default algorithms are probabilistic.

For an algorithm  $A$ ,  $y \leftarrow A(x)$  means that  $A$  is run on input  $x$  (with fresh random coins if  $A$  is probabilistic), and that the result is stored in  $y$ . More generally  $y \leftarrow a$  states that the result of the evaluation of the expression  $a$  is stored in the variable  $y$ .

An interactive Turing machine is a special kind of Turing machines able to communicate with external algorithms. To do so, the interactive Turing machine uses additional tapes to communicate with the other Turing machines, namely an input tape to send messages, and an output tape to receive messages. These other Turing machines are called the *oracles* of the interactive Turing machine. We write  $\mathcal{T}^{O_1, O_2}(x)$  to say that the Turing machine  $\mathcal{T}$  is called with input  $x$  and has access to the oracles  $O_1$  and  $O_2$ .

For a distribution  $\mathcal{D}$ ,  $A(\mathcal{D})$  denotes the output of the execution of  $A$  on an input  $x$  sampled from the distribution  $\mathcal{D}$ .

In this manuscript, we will often abuse notation, and identify a Turing machine and the algorithm it runs.

Finally, we will say that an algorithm is *efficient* if it runs in time polynomial in the size of its arguments.

**Protocols.** A two-party protocol  $P = (A_1, A_2)$  is a pair of algorithms  $A_1$  and  $A_2$ , interactively executed by a pair of two Turing machines  $\mathcal{T}_1$  and  $\mathcal{T}_2$ . We denote the execution of the protocol  $P$  as

$$P(\text{input}_1; \text{input}_2) = (A_1(\text{input}_1), A_2(\text{input}_2)),$$

meaning that  $A_1$  (resp.  $A_2$ ) is executed by  $\mathcal{T}_1$  (resp.  $\mathcal{T}_2$ ) with input  $\text{input}_1$  (resp.  $\text{input}_2$ ). We write

$$(\text{out}_1; \text{out}_2) \stackrel{\S}{\leftarrow} A_1(\text{input}_1) \leftrightarrow A_2(\text{input}_2)$$

to mean that  $\text{out}_1$  and  $\text{out}_2$  are the outputs of the interaction between  $A_1$  on input  $\text{input}_1$  and  $A_2$  on input  $\text{input}_2$ , respectively. We also simplify this notation and denote the result of the execution of  $P$  as

$$(\text{out}_1; \text{out}_2) \stackrel{\S}{\leftarrow} P(\text{input}_1; \text{input}_2).$$

In this formalism, we consider the messages  $\tau_{1 \rightarrow 2}$  (resp.  $\tau_{1 \leftarrow 2}$ ) sent by  $\mathcal{T}_1$  to  $\mathcal{T}_2$  (resp.  $\mathcal{T}_2$  to  $\mathcal{T}_1$ ) as part of the output  $\text{out}_1$  (resp.  $\text{out}_2$ ). These messages are called the *transcript* of  $\mathcal{T}_1$  (resp.  $\mathcal{T}_2$ ). Transcripts might be omitted from the output of the protocol for simplicity.

**Miscellaneous.** As mentioned earlier, the base-2 logarithm of the value  $x$  is  $\log x$ . When the variable  $T$  is a dictionary,  $T[v]$  denotes the item associated to  $v$ . If no item is associated to  $v$ , we write  $T[v] = \perp$ .

## 2.2 Cryptographic Preliminaries

### 2.2.1 Cryptographic Tools

**Security parameter.** In order to properly formalize security notions, we need to bound the computing power of an attacker. Indeed, one can always break cryptosystems using a large enough computer and spending a high amount of time. However, in cryptography, we restrict ourselves to the defence against *reasonable* attackers. To do so, we use the notion of *security parameter*, denoted  $\lambda \in \mathbb{N}$ . The security parameter is passed as an input to the attacker, under its unary representation  $1^\lambda$ , and we only consider attackers whose running time is polynomial in  $\lambda$ , and whose success probability is non-negligible in  $\lambda$ .

All these notions are formally defined in the following paragraphs.

**Adversaries.** An adversary is a probabilistic Turing machine, which, in this manuscript, run in polynomial time, which may carry a state when they need to be called several times. In most cases, we implicitly give as input to the adversary, both the unary representation of the security parameter, and the state. As adversaries' inputs are always polynomial in the security parameter, the polynomial time adversary runs in time polynomial in the security parameter.

**Games.** Security notions are often defined using security games (or experiments). Simple games are defined by having an adversary accessing a set of oracles, sometimes with some restrictions on calls to these oracles, and the output of the game is defined as the output of the adversary.

More generally, games are defined using the *code-based games* formalism introduced in [BR06]. Such a game  $G$  is a set of oracle procedures – including an initialization  $\text{Init}$  procedure and a finalization  $\text{Final}$  procedure – that is executed with an adversary  $A$ , *i.e.*  $A$  has access to the procedures, with some possible restrictions. For instance, the  $\text{Init}$  oracle is always the first one to be called and  $\text{Final}$  the last one, once  $A$  halted, taking  $A$ 's output as input. The output of  $\text{Final}$  is called the output of the game and is denoted  $G^A(1^\lambda)$ . When  $\text{Final}$  is omitted, it just forwards the adversary's output.

In those games, at startup, the boolean variables are initialized to false and the integer variables to 0.

**Statistical indistinguishability.** Let  $\mathcal{D}_1$  and  $\mathcal{D}_2$  be two distributions over the set  $S$ . Distributions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are said to be *statistically indistinguishable* if

$$\Delta(\mathcal{D}_1, \mathcal{D}_2) \leq \text{negl}(\lambda).$$

We denote

$$\mathcal{D}_1 \approx \mathcal{D}_2$$

the fact that  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are statistically indistinguishable.

**Computational indistinguishability.** Let  $\mathcal{D}_1$  and  $\mathcal{D}_2$  be two distributions which can be sampled in polynomial-time in  $\lambda$ , and  $A$  be a polynomial-time adversary  $A$  outputting a single bit. The *advantage of  $A$  distinguishing  $\mathcal{D}_1$  and  $\mathcal{D}_2$*  is defined by

$$\text{Adv}^{\mathcal{D}_1, \mathcal{D}_2}(A, 1^\lambda) = |\mathbb{P}[A(\mathcal{D}_1) = 1] - \mathbb{P}[A(\mathcal{D}_2) = 1]|.$$

Distributions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are said to be *computationally indistinguishable* if for any polynomial-time adversary  $A$ , the advantage of  $A$  in distinguishing  $\mathcal{D}_1$  and  $\mathcal{D}_2$  is negligible in  $\lambda$ . We denote

$$\mathcal{D}_1 \approx_c \mathcal{D}_2$$

the fact that  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are computationally indistinguishable. Note that two statistically indistinguishable distributions are computationally indistinguishable.

Similarly, we say that two different games  $G_0$  and  $G_1$ , both outputting one bit, are indistinguishable if, for any polynomial-time adversary  $A$ , the advantage of  $A$  in distinguishing  $G_0$  and  $G_1$ , denoted  $\text{Adv}^{G_0, G_1}(A, 1^\lambda)$  and defined as

$$\text{Adv}^{G_0, G_1}(A, 1^\lambda) = |\mathbb{P}[G_0^A = 1] - \mathbb{P}[G_1^A = 1]|,$$

is negligible in  $\lambda$ . In this case, we write  $G_0 \approx_c G_1$ .

**Game-based proofs.** Many of the security notions that we will define and use in this thesis are based on the indistinguishability of two different games  $G_0$  and  $G_1$ . Unfortunately, in many cases, we will not be able to directly prove this indistinguishability. Instead, we proceed by *game hops*, by constructing a sequence of games, starting with  $G_0$ , and ending with  $G_1$ , and proving that consecutive games are indistinguishable. The distinguishing advantage between  $G_0$  and  $G_1$  of an adversary  $A$  will then be the sum of the distinguishing advantages of  $A$  between every pair of consecutive games in the games sequence.

**The random oracle model (ROM).** The Random Oracle Model (or ROM), formally introduced by Bellare and Rogaway in [BR93], is a computational model where all parties have access to a (public) random oracle. As its name indicates, a random oracle outputs a random string for every new input it is given.

To prove the security of some schemes in the ROM, we often use an additional feature, called *programmability*. This feature allows the games for pre-programming the output of the random oracle on some inputs, in a way that the programmed random oracle is indistinguishable from a regular random oracle.

The ROM is a useful tool to show the security of some schemes. However, in practice, random oracles cannot exist (they would require an infinite description), and are often instantiated using hash functions. Actually, there is much debate among cryptographers on the quality of the ROM as an abstraction to analyze the security of cryptosystems [KM15]. Yet, for applied and real-world cryptography, it is a widely accepted and widely used model, as there is no convincing evidence that ROM-protocols have non-theoretical security weaknesses.

### 2.2.2 Hardness Assumptions

Cryptographic primitives rely on the hardness of some mathematical problems. We describe here the ones that will be useful for our constructions.

**The RSA assumption.** The RSA assumption, as introduced by Rivest, Shamir and Adleman in [RSA78] states that it is infeasible to compute the  $e$ -th root of an element modulo  $N$  when  $N$  is a product of two large primes, and  $e$  is relatively prime with  $\varphi(N)$ .

Let  $\text{RSAGen}$  be defined as the function, which, on input the security parameter  $1^\lambda$ , randomly samples two distinct  $\lambda$  bits primes  $p$  and  $q$ , computes  $N = p \cdot q$ , randomly picks an integer  $e$  less than

and relatively prime to  $\varphi(N) = (p-1)(q-1)$ , and outputs the pair  $(N, e)$ . For any adversary  $A$ , let the advantage of  $A$  in solving the RSA problem,  $\text{Adv}_A^{\text{RSA}}(\lambda)$ , be defined as:

$$\text{Adv}_A^{\text{RSA}}(\lambda) = \mathbb{P}[(N, e) \leftarrow \text{RSAGen}(1^\lambda), y \xleftarrow{\$} \mathbb{Z}_N^*, x \leftarrow A(1^\lambda, N, e, y) : x^e = y \pmod N].$$

The RSA assumption is that the RSA problem is hard: for any polynomial-time adversary  $A$ ,  $\text{Adv}_A^{\text{RSA}}(\lambda)$  is negligible in  $\lambda$ .

**Discrete logarithm.** Solving the discrete logarithm problem in the cyclic group  $\mathbb{G}$  with generator  $g$ , and of order  $N$  consists in finding the integer  $x \in \mathbb{Z}_N$  such that  $g^x = h$  for an element  $h \in \mathbb{G}$ .

In terms of security games, this can be formalized as follows. For any adversary  $A$ , let  $\text{Adv}_{\mathbb{G}, A}^{\text{DL}}(\lambda)$  be the quantity

$$\text{Adv}_{\mathbb{G}, A}^{\text{DL}}(\lambda) = \mathbb{P}[h \xleftarrow{\$} \mathbb{G}, x \leftarrow A(1^\lambda, \mathbb{G}, g, h) : g^x = h].$$

We say that the discrete logarithm is hard in  $\mathbb{G}$  if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\mathbb{G}, A}^{\text{DL}}(\lambda)$  is negligible in  $\lambda$ . The discrete log is believed to be hard on large prime order subgroups of  $(\mathbb{F}_p^*, \times)$ , and on cyclic subgroups of elliptic curves over finite fields: this is the discrete logarithm assumption.

**The Diffie-Hellman assumptions.** A strengthening of the discrete logarithm assumption is the Computational Diffie-Hellman (CDH) assumption. It requires that an adversary, given  $g^a$  and  $g^b$ , for  $g$  a generator of the group  $\mathbb{G}$  of order  $N$ , and  $a, b \in \mathbb{Z}_n$ , cannot efficiently compute  $g^{a \cdot b}$ . Formally, for an adversary  $A$ , we define the advantage  $\text{Adv}_{\mathbb{G}, A}^{\text{CDH}}(\lambda)$  as

$$\text{Adv}_{\mathbb{G}, A}^{\text{CDH}}(\lambda) = \mathbb{P}[a \xleftarrow{\$} \mathbb{Z}_N, b \xleftarrow{\$} \mathbb{Z}_N, h \leftarrow A(1^\lambda, \mathbb{G}, g^a, g^b) : g^{ab} = h].$$

We say that the CDH assumption is hard in  $\mathbb{G}$  if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\mathbb{G}, A}^{\text{CDH}}(\lambda)$  is negligible in  $\lambda$ . The CDH assumption is supposed to be hard on large prime order subgroups of  $(\mathbb{F}_p^*, \times)$ , and on cyclic subgroups of elliptic curves over finite fields.

A stronger assumption is also commonly encountered, the decisional version of CDH, called the Decisional Diffie-Hellman (DDH) assumption. This time the adversary is asked to distinguish between the triple  $(g^a, g^b, g^{a \cdot b})$  and the triple  $(g^a, g^b, g^c)$ , where  $c$  is picked randomly in  $\mathbb{Z}_n$ .

$$\text{Adv}_{\mathbb{G}, A}^{\text{DDH}}(\lambda) = \left| \mathbb{P}[(a, b) \xleftarrow{\$} \mathbb{Z}_N^2 : A(1^\lambda, g^a, g^b, g^{ab}) = 1] - \mathbb{P}[(a, b, z) \xleftarrow{\$} \mathbb{Z}_N^3 : A(1^\lambda, g^a, g^b, g^z) = 1] \right|.$$

We say that the DDH assumption is hard in  $\mathbb{G}$  if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\mathbb{G}, A}^{\text{DDH}}(\lambda)$  is negligible in  $\lambda$ . The DDH assumption is also supposed to be hard on large prime order subgroups of  $(\mathbb{F}_p^*, \times)$ , and on cyclic subgroups of elliptic curves over finite fields.

**Cryptographic pairings.** We will require that bilinear groups satisfy a hardness assumption called the Decisional Bilinear Diffie-Hellman (DBDH) assumption [BB04]: it requires that a bounded adversary cannot distinguish the tuple  $(g, g^a, g^b, g^c, e(g, g)^{abc})$  from the tuple  $(g, g^a, g^b, g^c, e(g, g)^z)$ , where  $a, b, c$  and  $z$  are randomly generated.



Formally, for a bilinear group  $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ , the advantage  $\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, A}^{\text{DBDH}}(\lambda)$  of an adversary  $A$  in the Decisional Bilinear Diffie-Hellman game is defined as:

$$\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, A}^{\text{DBDH}}(\lambda) = \left| \mathbb{P}[(a, b, c) \xleftarrow{\$} \mathbb{Z}_N^3 : A(1^\lambda, g^a, g^b, g^c, e(g, g)^{abc}) = 1] - \mathbb{P}[(a, b, c, z) \xleftarrow{\$} \mathbb{Z}_N^4 : A(1^\lambda, g^a, g^b, g^c, e(g, g)^z) = 1] \right|.$$

The bilinear group is said to be secure if, for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, A}^{\text{DBDH}}(\lambda)$  is negligible in  $\lambda$ .

In this definition, we only considered the symmetric setting for the bilinear group, while the definition can trivially be adapted to an asymmetric pairing. In practice, cryptographic pairings are instantiated using elliptic curves, and we will use Type-3 pairings only. We refer to [GPS06] for more details on pairings.

## 2.3 Cryptographic Primitives

In this Section, we define and describe the cryptographic primitives that we will use throughout this thesis. Completely formal definitions of most of these objects can be found in [Gol04] (we often adopt here a simplified formulation).

### 2.3.1 Pseudorandom Function (PRF)

A pseudorandom function is a function that is computationally indistinguishable from a truly random function. More formally, let  $F : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$  be a polynomial-time computable map, where  $\mathcal{K}$  and  $\mathcal{R}$  are finite.  $\mathcal{K}$  is the *key space* of  $F$ , and  $\mathcal{D}$  its domain, while  $\mathcal{R}$  is the range of  $F$ . They respectively have size  $|\mathcal{K}| = 2^{\ell_{\mathcal{K}}(\lambda)}$ ,  $|\mathcal{D}| = 2^{\ell_{\mathcal{D}}(\lambda)}$ , and  $|\mathcal{R}| = 2^{\ell_{\mathcal{R}}(\lambda)}$ , with  $\ell_{\mathcal{K}}, \ell_{\mathcal{D}}, \ell_{\mathcal{R}} : \mathbb{N} \rightarrow \mathbb{N}$ . For  $K \in \mathcal{K}$ , we denote  $F_K$  the function that is the partial evaluation of  $F$  on  $K$ , namely

$$\begin{aligned} F_K : \mathcal{D} &\rightarrow \mathcal{R} \\ x &\mapsto F(K, x) \end{aligned}$$

Hence,  $F$  can be seen as a *function family*.

**Definition 2.1** (Pseudorandom function). *Let  $F : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$  be a function family defined as above, and  $\text{Func}(\mathcal{D}, \mathcal{R})$  the set of the functions of domain  $\mathcal{D}$  and range  $\mathcal{R}$ . The pseudorandom function distinguishing advantage  $\text{Adv}_{A, F}^{\text{prf}}(\lambda)$  of  $A$  against  $F$  is defined as*

$$\text{Adv}_{F, A}^{\text{prf}}(\lambda) = \left| \mathbb{P}[K \xleftarrow{\$} \mathcal{K} : A^{F_K(\cdot)}(1^\lambda) = 1] - \mathbb{P}[\pi \xleftarrow{\$} \text{Func}(\mathcal{D}, \mathcal{R}) : A^{\pi(\cdot)}(1^\lambda) = 1] \right|.$$

The PRF advantage function of  $F$  is defined as follows. For any integers  $t, q$ ,

$$\text{Adv}_F^{\text{prf}}(\lambda, t, q) = \max_A \text{Adv}_{F, A}^{\text{prf}}(\lambda)$$

where the maximum is taken over all adversary  $A$  with time complexity  $t$ , making at most  $q$  oracle queries. The function  $F$  is said to be a pseudorandom function if  $\text{Adv}_{F, A}^{\text{prf}}(\lambda)$  is negligible in  $\lambda$  for any polynomial-time adversary  $A$ .

### 2.3.2 Constrained Pseudorandom Function (CPRF)

The idea of *constrained PRFs* (CPRF) has been introduced in independent works by Boneh and Waters, Boyle *et al.*, and Kiayias *et al.* [BW13; BGI14; KPTZ13]. A constrained PRF is associated with a family of boolean circuits  $\mathcal{C}$ . The holder of the master PRF key is able to compute a *constrained key*  $K_C$  corresponding to a circuit  $C \in \mathcal{C}$ ; the constrained key  $K_C$  allows for evaluation of the PRF on inputs  $x$  such that  $C(x) = 1$ , but only on the inputs satisfying this predicate.

More formally, a *constrained PRF*  $F$  with respect to a circuit family  $\mathcal{C}$  is a mapping  $F : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$ , together with a pair of algorithms  $(F.\text{Constrain}, F.\text{Eval})$ , defined as follows.

- $F.\text{Constrain}(K, C)$  is a probabilistic polynomial-time algorithm taking as input a key  $K \in \mathcal{K}$  and a circuit  $C \in \mathcal{C}$ . It outputs a constrained key  $K_C$ .
- $F.\text{Eval}(K_C, x)$  is a deterministic polynomial-time algorithm taking as input a constrained key  $K_C$  for circuit  $C$ , and  $x \in \mathcal{D}$ . It outputs is an element  $y \in \mathcal{R}$ .

When no confusion can arise, we may leave out Eval and write  $F.\text{Eval}(K_C, x)$  as  $F(K_C, x)$ . As for the regular PRF case, the key space, domain and range have respective size  $|\mathcal{K}| = 2^{\ell_{\mathcal{K}}(\lambda)}$ ,  $|\mathcal{D}| = 2^{\ell_{\mathcal{D}}(\lambda)}$ , and  $|\mathcal{R}| = 2^{\ell_{\mathcal{R}}(\lambda)}$ , with  $\ell_{\mathcal{K}}, \ell_{\mathcal{D}}, \ell_{\mathcal{R}} : \mathbb{N} \rightarrow \mathbb{N}$ .

**Correctness.** A CPRF  $F$  is *correct* if  $C(x) = 1$  implies  $F(K, x) = F.\text{Eval}(K_C, x)$ , where  $K_C = F.\text{Constrain}(K, C)$ , for all  $K \in \mathcal{K}$ ,  $x \in \mathcal{D}$ , and  $C \in \mathcal{C}$ .

**Security Definition.** The security properties of a constrained PRF can be formalized using the security game  $G_{\text{cprf}}$  described in Figure 2.1. Informally, the adversary  $A$  wins the game (the game outputs 1) when he is able to distinguish between real evaluations of  $F$  and truly random elements of  $\mathcal{R}$  on inputs such that he never queried a constrained key  $K_C$  for a circuit  $C$  evaluating to 1 on these inputs. The formal definition follows.

<pre> Init()   <math>K \xleftarrow{\\$} \mathcal{K}</math>   <math>b \xleftarrow{\\$} \{0, 1\}</math>   <math>E \leftarrow \emptyset, Z \leftarrow \emptyset, L \leftarrow \emptyset</math> Challenge(<math>x</math>)   <math>Z \leftarrow Z \cup \{x\}</math>   <b>if</b> <math>b = 0</math> <b>then</b>     <math>y \xleftarrow{\\$} \mathcal{R}</math>   <b>else</b>     <math>y \leftarrow F(K, x)</math>   <b>end if</b>   <b>return</b> <math>y</math> </pre>	<pre> Eval(<math>x</math>)   <math>E \leftarrow E \cup \{x\}</math>   <b>return</b> <math>F(K, x)</math> Constrain(<math>C</math>)   <math>L \leftarrow L \cup C</math>   <b>return</b> <math>F.\text{Constrain}(C)</math> Final(<math>b'</math>)   <b>if</b> <math>b = b', E \cap Z = \emptyset</math>     <b>and</b> <math>\forall (C, z) \in (L, Z), C(z) = 0</math>     <b>return</b> 1 <span style="float: right;">▷ The adversary wins</span>   <b>return</b> 0 <span style="float: right;">▷ The adversary loses</span> </pre>
---	---

**Figure 2.1** – Procedures of the  $G_{\text{cprf}}$  security game. The lists  $E$ ,  $Z$ , and  $L$  are, respectively, the list of evaluated inputs, challenged inputs and constraints. The condition in Final ensures that the game is only challenged on constrained inputs, and never on an evaluated input.

**Definition 2.2** (Constrained PRF). Let  $F$  be a constrained function as defined previously. The advantage of the adversary  $A$  in the constrained PRF security game,  $\text{Adv}_{F,A}^{\text{cprf}}(\lambda)$ , is

$$\text{Adv}_{F,A}^{\text{cprf}}(\lambda) = \mathbb{P}[G_{\text{cprf}}^A(1^\lambda) = 1].$$

We say that  $F$  is a constrained pseudorandom function if, for any polynomial-time adversary  $A$ ,  $\text{Adv}_{F,A}^{\text{cprf}}(\lambda)$  is negligible in the security parameter  $\lambda$ .

### 2.3.3 Pseudorandom Permutation (PRP)

A pseudorandom permutation is a permutation that is indistinguishable from a truly random permutation. Let  $F : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$  be a function family. We say that  $F$  is a family of permutation if for every  $K \in \mathcal{K}$ ,  $F_K$  is a bijection between  $\mathcal{D}$  and  $\mathcal{R}$ . We will always be in the case where  $\mathcal{D} = \mathcal{R}$ .

**Definition 2.3** (Pseudorandom permutation). Let  $F : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{D}$  be a permutation family, and  $\text{Perm}(\mathcal{D})$  the set of all permutations of  $\mathcal{D}$ . The pseudorandom function distinguishing advantage  $\text{Adv}_{F,A}^{\text{prp}}(\lambda)$  of  $A$  against  $F$  is:

$$\text{Adv}_{F,A}^{\text{prp}}(\lambda) = \left| \mathbb{P}[K \xleftarrow{\$} \mathcal{K} : A^{F_K(\cdot)}(1^\lambda) = 1] - \mathbb{P}[\pi \xleftarrow{\$} \text{Perm}(\mathcal{D}) : A^{\pi(\cdot)}(1^\lambda) = 1] \right|.$$

The PRF advantage function of  $F$  is defined as follows. For any integers  $t, q$ ,

$$\text{Adv}_F^{\text{prp}}(\lambda, t, q) = \max_A \text{Adv}_{F,A}^{\text{prp}}(\lambda)$$

where the maximum is taken over all adversary  $A$  with time complexity  $t$ , making at most  $q$  oracle queries. The function  $F$  is said to be a pseudorandom permutation if  $\text{Adv}_{F,A}^{\text{prp}}(\lambda)$  is negligible in  $\lambda$  for any polynomial-time adversary  $A$ .

**PRF switching lemma.** It will be useful in the security proofs to be able to switch from a PRP to a PRF. To do so, we will use the well-known PRF switching lemma that states that a PRP is also a PRF. We refer the reader to [BR06, Lemma 1] for the proof.

**Lemma 2.1.** Let  $F$  be a PRP over the set  $\mathcal{D}$ . For any adversary  $A$  making at most  $q$  queries,

$$\left| \text{Adv}_{F,A}^{\text{prf}}(\lambda) - \text{Adv}_{F,A}^{\text{prp}}(\lambda) \right| \leq \frac{q^2}{2|\mathcal{D}|}.$$

### 2.3.4 Trapdoor Permutation (TDP)

Informally, a trapdoor permutation (TDP)  $\pi$  is a permutation over a set  $\mathcal{M}$  such that, using a public key PK,  $\pi$  can be easily evaluated, but the inverse  $\pi^{-1}$  can be efficiently computed only with the secret SK.

More formally, a family of trapdoor permutations over a set  $\mathcal{M}$  is a triple  $\pi$  of algorithms (KeyGen, Eval, Invert) such that:

- KeyGen is a randomized algorithm taking as input the security parameter  $1^\lambda$  that generates a pair (SK, PK), where SK is the private key and PK the public key;
- Eval is a deterministic polynomial-time algorithm taking as inputs a public key and an element in  $\mathcal{M}$ , and such that for every public key PK generated by KeyGen,  $\pi(\text{PK}, \cdot)$  is a bijection over  $\mathcal{M}$ ;

- Invert is a deterministic polynomial-time algorithm taking as inputs a secret key and an element in  $\mathcal{M}$ , such that for every key pair  $(\text{PK}, \text{SK})$  generated by  $\text{KeyGen}$ ,

$$\text{Invert}(\text{SK}, \text{Eval}(\text{PK}, x)) = x.$$

In the following, we will simplify the notations, and use  $\pi_{\text{PK}}(\cdot)$  to denote  $\text{Eval}(\text{PK}, \cdot)$  and  $\pi_{\text{SK}}^{-1}(\cdot)$  for  $\text{Invert}(\text{SK}, \cdot)$ .

**Definition 2.4** (Secure trapdoor permutation). *For an adversary  $A$ , the advantage  $\text{Adv}_{\pi, A}^{\text{tdp}}(\lambda)$  of  $A$  in the trapdoor permutation security game is:*

$$\text{Adv}_{\pi, A}^{\text{tdp}}(\lambda) = \Pr[y \xleftarrow{\$} \mathcal{M}, (\text{SK}, \text{PK}) \leftarrow \text{KeyGen}(1^\lambda), x \leftarrow A(1^\lambda, \text{PK}, y) : \pi_{\text{PK}}(x) = y].$$

For any integer  $t$ ,  $\text{Adv}_{\pi}^{\text{tdp}}(\lambda, t)$  is:

$$\text{Adv}_{\pi}^{\text{tdp}}(\lambda, t) = \max_A \text{Adv}_{\pi, A}^{\text{tdp}}(\lambda)$$

where the maximum is taken over all adversary  $A$  with time complexity  $t$ . The trapdoor permutation  $\pi$  is said to be secure if, for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\pi, A}^{\text{OW}}(\lambda)$  is negligible in  $\lambda$ .

We use the notation  $\pi_{\text{PK}}^{(c)}(x)$  (resp.  $\pi_{\text{SK}}^{(-c)}(x)$ ) for the iterated application of  $\pi_{\text{PK}}$  (resp.  $\pi_{\text{SK}}^{-1}$ )  $c$  times.

### 2.3.5 Hash Function

A hash function family is a polynomial-time computable map  $H : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$  where  $\mathcal{K}$  and  $\mathcal{R}$  are non-empty sets. We denote the partial evaluation of  $H$  on  $K$  as  $H_K(\cdot)$ . For hash functions, we are interested in the difficulty with which an adversary is able to find two distinct elements in  $\mathcal{D}$  evaluating to the same elements in  $\mathcal{R}$ .

**Definition 2.5.** *Let  $H : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$  be a hash function family, and for any adversary  $A$ , let*

$$\text{Adv}_{H, A}^{\text{col}}(\lambda) = \mathbb{P}[K \xleftarrow{\$} \mathcal{K}, (M, M') \leftarrow A(K) : M \neq M' \wedge H_K(M) = H_K(M')].$$

*The hash function family  $H$  is said to be a collision-resistant family of hash function if, for any polynomial-time adversary  $A$ ,  $\text{Adv}_{H, A}^{\text{col}}(\lambda)$  is negligible in  $\lambda$ .*

In practice, we only have access to a single element of the hash function family, which is denoted  $H$ .

**Instantiating the ROM.** Frequently, the random oracle used in the ROM (cf. Section 2.2.1) will be instantiated using a hash function. Unfortunately, many hash functions share undesirable properties (e.g. length-extension attacks) that make them unfit for such direct use as a random oracle. Instead, we will use the HMAC construction [BCK96] with a public key as a random oracle. For a hash function  $H$ , HMAC is defined as

$$\text{HMAC}(K, x) = H((K \oplus \text{opad}) || H((K \oplus \text{ipad}) || x))$$

where  $\text{opad}$  and  $\text{ipad}$  are two constants and  $\oplus$  is the exclusive-OR (XOR) operation.

**(Multi)set hashing.** In the case  $\mathcal{D}$  is a set of sets, it would be nice to be able to easily compute the hash of  $S \cup S'$  from the hashes of  $S$  and  $S'$ . Multiset hashing was introduced by Clarke *et al.* [CDD+03], based on the framework proposed by Bellare and Micciancio [BM97] for incremental hashing. Here, we slightly extend their definition so it fits our needs in this thesis.

A *set hashing function* is a family of quadruples of probabilistic polynomial algorithms  $(\mathcal{H}_K, \equiv_{\mathcal{H}_K}, +_{\mathcal{H}_K}, -_{\mathcal{H}_K})$ , indexed by a key  $K \in \mathcal{K}$ , such that  $\mathcal{H}_K : \mathcal{P}(\mathcal{D}_K) \rightarrow \mathcal{R}_K$  maps sets whose elements are in  $\mathcal{D}_K$  to an element in  $\mathcal{R}_K$ , and for all  $S \subset \mathcal{P}(\mathcal{D}_K)$ ,

- $\mathcal{H}_K(S) \equiv_{\mathcal{H}_K} \mathcal{H}_K(S)$  (comparability)
- $\forall x \in \mathcal{D} \setminus S, \mathcal{H}_K(S \cup \{x\}) \equiv_{\mathcal{H}_K} \mathcal{H}_K(S) +_{\mathcal{H}_K} \mathcal{H}_K(\{x\})$  (insertion incrementality)
- $\forall x \in S, \mathcal{H}_K(S \setminus \{x\}) \equiv_{\mathcal{H}_K} \mathcal{H}_K(S) -_{\mathcal{H}_K} \mathcal{H}_K(\{x\})$  (deletion incrementality).

We want multiset hash functions to be secure in the sense of collision resistance as a regular hash function: it is infeasible for an adversary to find two sets hashing to the same value. Note that in this definition, we chose to make the domain and range to directly depend on the key  $K$  (previously, they depended only on its size).

**Definition 2.6.** Let  $\mathcal{H}$  be a set hashing function. For any adversary  $A$ , let

$$\text{Adv}_{\mathcal{H}, A}^{\text{col}}(\lambda) = \mathbb{P}[K \xleftarrow{\$} \mathcal{K}, (S, S') \leftarrow A(K) : S \neq S' \wedge \mathcal{H}_K(S) \equiv_{\mathcal{H}_K} \mathcal{H}_K(S')].$$

The MSet-Mu-Hash construction of Clark *et al.* [CDD+03] for multisets works as follows: if  $H$  is a regular (*i.e.* non incremental) hash function,  $\mathcal{H}(x_1^{m_1}, \dots, x_n^{m_n})$  is defined as  $\prod H(x_i)^{m_i}$ , where  $x_i^{m_i}$  represents the element  $x_i$  with multiplicity  $m_i$ . Formally, MSet-Mu-Hash is defined as follows:

$$\begin{aligned} \mathcal{H}_K(M) : \mathcal{P}(\mathcal{D}) &\rightarrow \mathbb{F}_q \\ M &\mapsto \prod_{x \in \mathcal{D}} H_{K_H}(x)^{M_x} \end{aligned}$$

where  $H : \mathcal{D} \rightarrow \mathbb{F}_q$  is a hash function from the set  $\mathcal{D}$  to the field  $\mathbb{F}_q$  indexed by the key  $(K_H, q) \in \mathcal{K}_H$  where  $q$  is a prime of size  $\text{poly}(\lambda)$ , and  $M_x$  is the multiplicity of  $x$  in  $M$ . The key  $K$  of  $\mathcal{H}$  is then defined as  $(K_H, q)$ . This construction clearly fits our functional needs: for  $S \subset \mathcal{D}$ , we can easily compute (*i.e.* in constant time)  $\mathcal{H}_K(S \cup \{x\})$  (resp.  $\mathcal{H}_K(S \setminus \{x\})$ ) from  $\mathcal{H}_K(S)$  and  $\mathcal{H}_K(\{x\}) = H(x)$  – or even from  $x$  if we have access to  $H$  – as  $\mathcal{H}_K(S \cup \{x\}) = \mathcal{H}_K(S) \cdot H_{K_H}(x)$  (resp.  $\mathcal{H}_K(S \setminus \{x\}) = \mathcal{H}_K(S) \cdot H_{K_H}(x)^{-1}$ ).

Clarke *et al.* show that  $\mathcal{H}$  is collision resistant as long as the discrete log assumption holds in  $\mathbb{F}_q$  when  $H$  is modeled as a random oracle.

**Theorem 2.2** (Theorem 2 of [CDD+03]). *If the discrete log assumption holds in  $\mathbb{F}_q$ , and  $H$  is a (non-programmable) random oracle, the multiset hash function  $\mathcal{H}$  is collision resistant.*

Note that multiset hashing can also be based on elliptic curves for improved efficiency [MTA16]. The security of the construction would immediately follow, using the hardness of discrete logarithm on cyclic subgroups of elliptic curves instead of its hardness on finite fields.

Note that, as for regular hash functions, we will omit the key as, in practice, we only have access to a single element of the multiset hash function family.

### 2.3.6 Semantically Secure Encryption

A (symmetric) encryption scheme  $SE$  is a triple of algorithms (KeyGen, Enc, Dec). The randomized key generation algorithm KeyGen takes as input the security parameter in its unary form and outputs a key  $K$  from the key set  $\mathcal{K}$ . The encryption algorithm Enc takes a key and a plaintext  $m$  in the message space  $\mathcal{M}$  and outputs a ciphertext  $c \leftarrow \text{Enc}(K, m)$  from the ciphertext space  $\mathcal{C}$ . Note that Enc can be either randomized or deterministic. The decryption algorithm is deterministic, and as input a key  $K$  and a string  $c$ , and outputs either an element  $m \in \mathcal{M}$  or the symbol  $\perp$ .

In the following, we will only consider *correct* schemes, that is schemes such that, for all keys  $K \in \mathcal{K}$ , and all messages  $m \in \mathcal{M}$ ,

$$\text{Dec}(K, \text{Enc}(K, m)) = m.$$

Many security definitions have been developed for (symmetric) encryption. Here we will consider indistinguishability against chosen plaintext attacks (IND-CPA). More precisely, we use the Left-Or-Right (LOR-CPA) definition, as given by Bellare *et al.* [BDJR97].

**Definition 2.7.** Let  $SE = (\text{KeyGen}, \text{Enc}, \text{Dec})$  be a symmetric encryption scheme. For  $b \in \{0, 1\}$ , we define LoR as

$$\text{LoR}(x_0, x_1, b) = x_b.$$

For an adversary  $A$ , the IND-CPA advantage of  $A$  against  $SE$  is

$$\text{Adv}_{SE,A}^{\text{cpa}}(\lambda) = \left| \mathbb{P}[K \xleftarrow{\$} \text{KeyGen}(1^\lambda) : A^{\text{Enc}_K(\text{LoR}(\cdot, \cdot, 0))}(1^\lambda) = 1] - \mathbb{P}[K \xleftarrow{\$} \text{KeyGen}(1^\lambda) : A^{\text{Enc}_K(\text{LoR}(\cdot, \cdot, 1))}(1^\lambda) = 1] \right|,$$

with the restriction that  $A$  must only query the oracle  $\text{Enc}_K(\text{LoR}(\cdot, \cdot, b))$  with pairs of messages of equal length. The IND-CPA advantage function of  $SE$  is defined as follows. For any integers  $t, q, \mu$ ,

$$\text{Adv}_{SE}^{\text{cpa}}(\lambda, t, q, \mu) = \max_A \text{Adv}_{SE,A}^{\text{cpa}}(\lambda)$$

where the maximum is taken over all adversary  $A$  with time complexity  $t$ , making at most  $q$  oracle queries on messages of total length at most  $\mu$ .  $SE$  is said to be a IND-CPA-secure if  $\text{Adv}_{SE,A}^{\text{cpa}}(\lambda)$  is negligible in  $\lambda$  for any polynomial-time adversary  $A$ .

In practice, we will suppose that  $\mathcal{K} = \{0, 1\}^\lambda$ , and  $\mathcal{M} = \mathcal{C} = \{0, 1\}^*$ , unless otherwise specified. Also, the KeyGen algorithm will just pick a key in  $\mathcal{K}$  uniformly at random.

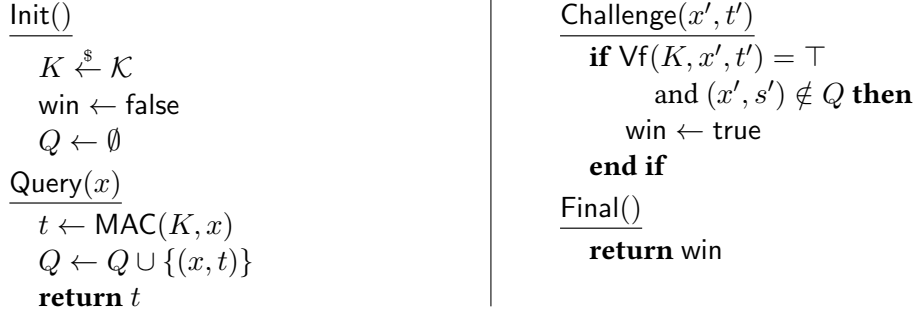
### 2.3.7 Message Authentication Code (MAC)

A message authentication code is used to ensure that a message comes from the right sender. It is a triple of algorithms (KeyGen, MAC, Vf). The randomized key generation algorithm KeyGen takes as input the security parameter in its unary form and outputs a key  $K$  from the key set  $\mathcal{K}$ . The algorithm MAC takes as input  $K \in \mathcal{K}$  and a string  $m \in \{0, 1\}^*$  and outputs a tag  $T \in \{0, 1\}^\lambda$ . Finally Vf, on input a key  $K$ , a string  $m$  and a tag  $T$ , outputs  $\perp$  or  $\top$ .

We require that a MAC is correct, namely that for all  $K \in \mathcal{K}$ , and all string  $m \in \{0, 1\}^*$ ,

$$\text{Vf}(K, m, \text{MAC}(K, m)) = \top.$$

The security requirement of a MAC will be that it is infeasible for any polynomial-time adversary to forge a valid tag without the secret key.



**Figure 2.2** – Procedures of the  $G_{\text{euf-cma}}$  security game.

**Definition 2.8.** Let  $(\text{KeyGen}, \text{MAC}, \text{Vf})$  be a message authentication code. The advantage of  $A$  in the existential unforgeability with chosen messages attack game (EUF-CMA),  $\text{Adv}_{A, \text{MAC}}^{\text{euf-cma}}(\lambda)$ , is

$$\text{Adv}_{\text{MAC}, A}^{\text{euf-cma}}(\lambda) = \mathbb{P}[G_{\text{euf-cma}}^A(1^\lambda) = 1].$$

where the  $G_{\text{euf-cma}}$  is described in Figure 2.2. The EUF-CMA advantage function is defined as follows. For any integers  $t, q, \mu$ ,

$$\text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\lambda, t, q, \mu) = \max_A \text{Adv}_{\text{MAC}, A}^{\text{euf-cma}}(\lambda)$$

where the maximum is taken over all adversary  $A$  with time complexity  $t$ , making at most  $q$  oracle queries on messages of total length at most  $\mu$ . MAC is said to be an EUF-CMA-secure MAC if  $\text{Adv}_{\text{MAC}, A}^{\text{euf-cma}}(\lambda)$  is negligible in  $\lambda$  for any polynomial-time adversary  $A$ .

The most handy and practical way to instantiate a MAC is to use a PRF with variable input length, i.e. with domain  $\mathcal{D}$ . For such a PRF  $F$ , we will define MAC and Vf as

$$\begin{aligned} \text{MAC}(K, x) &= F(K, x) \\ \text{Vf}(K, x, t) &= \begin{cases} \top & \text{if } F(K, x) = t \\ \perp & \text{otherwise.} \end{cases} \end{aligned}$$

### 2.3.8 Authenticated Encryption with Associated Data (AEAD)

Using authenticated encryption with associated, one is able to ensure both the confidentiality of a message and the authenticity of the message plus some optional additional data. An AEAD scheme  $SE$  is a triple of algorithms  $(\text{KeyGen}, \text{Enc}, \text{Dec})$ .

The randomized key generation algorithm  $\text{KeyGen}$  takes as input the security parameter in its unary form and outputs a key  $K$  from the key set  $\mathcal{K}$ . The encryption algorithm  $\text{Enc}$  takes a key, an optional string  $a$  called additional data, and a plaintext  $m$  in the message space  $\mathcal{M}$  and outputs a ciphertext  $c \leftarrow \text{Enc}(K, a, m)$  from the ciphertext space  $\mathcal{C}$ . The decryption algorithm is deterministic, and as input a key  $K$ , the (optional) additional data  $a$ , and a string  $c$ , and outputs either an element  $m \in \mathcal{M}$  or the symbol  $\perp$ .

In the following, we will only consider *correct* schemes, that is schemes such that, for all keys  $K \in \mathcal{K}$ , all string  $a \in \{0, 1\}^*$ , and all messages  $m \in \mathcal{M}$ ,

$$\text{Dec}(K, a, \text{Enc}(K, a, m)) = m.$$

**Definition 2.9.** Let  $SE = (\text{KeyGen}, \text{Enc}, \text{Dec})$  be an authenticated encryption scheme with additional data. For an adversary  $A$ , the AE advantage of  $A$  against  $SE$  is

$$\text{Adv}_{SE,A}^{\text{ae}}(\lambda) = \left| \mathbb{P}[K \stackrel{\$}{\leftarrow} \text{KeyGen}(1^\lambda) : A^{\text{Enc}_K(\cdot, \cdot), \text{Dec}_K(\cdot, \cdot)}(1^\lambda) = 1] - \mathbb{P}[A^{\$(\cdot, \cdot), \perp(\cdot, \cdot)}(1^\lambda) = 1] \right|$$

where  $\$$  is an oracle that, on input  $(a, m)$ , picks a random string  $r$  of size  $|m|$  and returns  $\text{Enc}_K(a, r)$  and  $\perp$  is the oracle always returning the symbol  $\perp$ , with the restriction that  $\text{Dec}$  (resp.  $\perp$ ) is never called on a output of  $\text{Enc}$  (resp.  $\$$ ).

The AE advantage function of  $SE$  is defined as follows. For any integers  $t, q_e, \mu_e, q_d, \mu_d$ ,

$$\text{Adv}_{SE}^{\text{ae}}(\lambda, t, q_e, \mu_e, q_d, \mu_d) = \max_A \text{Adv}_{SE,A}^{\text{ae}}(\lambda)$$

where the maximum is taken over all adversary  $A$  with time complexity  $t$ , making at most  $q_e$  (resp.  $q_d$ ) encryption (resp. decryption) oracle queries on messages of total length at most  $\mu_e$  (resp.  $\mu_d$ ).  $SE$  is said to be a secure AEAD if  $\text{Adv}_{SE,A}^{\text{ae}}(\lambda)$  is negligible in  $\lambda$  for any polynomial-time adversary  $A$ .

Again, in practice, we will suppose that  $\mathcal{K} = \{0, 1\}^\lambda$ , and  $\mathcal{M} = \mathcal{C} = \{0, 1\}^*$ , unless otherwise specified, and that the  $\text{KeyGen}$  algorithm will just pick a key in  $\mathcal{K}$  uniformly at random.

## References

- [BB04] Dan Boneh and Xavier Boyen. *Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles*. In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 223–238 (cit. on p. 28).
- [BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *Keying Hash Functions for Message Authentication*. In: *CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 1–15 (cit. on p. 32).
- [BDJR97] Mihir Bellare, Anand Desai, Eric Jorjani, and Phillip Rogaway. *A Concrete Security Treatment of Symmetric Encryption*. In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 394–403 (cit. on p. 34).
- [BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. *Functional Signatures and Pseudorandom Functions*. In: *PKC 2014*. Ed. by Hugo Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 501–519 (cit. on p. 30).
- [BM97] Mihir Bellare and Daniele Micciancio. *A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost*. In: *EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, Heidelberg, May 1997, pp. 163–192 (cit. on p. 33).
- [BR06] Mihir Bellare and Phillip Rogaway. *The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs*. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 409–426 (cit. on pp. 26, 31).
- [BR93] Mihir Bellare and Phillip Rogaway. *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*. In: *ACM CCS 93*. Ed. by V. Ashby. ACM Press, Nov. 1993, pp. 62–73 (cit. on p. 27).
- [BW13] Dan Boneh and Brent Waters. *Constrained Pseudorandom Functions and Their Applications*. In: *ASIACRYPT 2013, Part II*. Ed. by Kazue Sako and Palash Sarkar. Vol. 8270. LNCS. Springer, Heidelberg, Dec. 2013, pp. 280–300 (cit. on p. 30).



- [CDD+03] Dwaine E. Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. *Incremental Multiset Hash Functions and Their Application to Memory Integrity Checking*. In: *ASIACRYPT 2003*. Ed. by Chi-Sung Laih. Vol. 2894. LNCS. Springer, Heidelberg, Nov. 2003, pp. 188–207 (cit. on p. 33).
- [Gol04] Oded Goldreich. *Foundations of cryptography*. Cambridge University Press, 2004 (cit. on pp. 29, 44, 47).
- [GPS06] S.D. Galbraith, K.G. Paterson, and N.P. Smart. *Pairings for Cryptographers*. Cryptology ePrint Archive, Report 2006/165. <http://eprint.iacr.org/2006/165>. 2006 (cit. on pp. 29, 115).
- [KM15] Neal Koblitz and Alfred Menezes. *The Random Oracle Model: A Twenty-Year Retrospective*. Cryptology ePrint Archive, Report 2015/140. <http://eprint.iacr.org/2015/140>. 2015 (cit. on p. 27).
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. *Delegatable pseudorandom functions and applications*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 669–684 (cit. on pp. 30, 87).
- [MTA16] Jeremy Maitin-Shepard, Mehdi Tibouchi, and Diego F Aranha. “Elliptic Curve Multiset Hash”. In: *The Computer Journal* 60.4 (2016), pp. 476–490 (cit. on pp. 33, 129).
- [RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on pp. 6, 27).

Encryption threatens to lead all of us to a very dark place.

JAMES COMEY

# Basics of Searchable Encryption 3

**F**ROM THE TOOLS INTRODUCED IN THE PREVIOUS CHAPTER, we can start to formalize the requirements of searchable encryption, especially in terms of security. In this chapter, we will start to formalize the problem and give security definitions, which will be used throughout this thesis. We will also introduce the notion of leakage, *i.e.* the information that the server is allowed to learn about the database and the queries. In particular, we will show that it is impossible to achieve both best possible security and optimal efficiency by proving two lower bounds. In a third section, we will present some results about the locality of single-keyword searchable encryption, and see that we cannot construct a secure scheme with the same performance as a plain database. Finally, we will quickly study some practical attacks against searchable encryption schemes, and see how important it is to understand the extent of the information that leaks from these constructions.

## Contents

---

<b>3.1</b>	<b>Definitions</b>	<b>40</b>
3.1.1	Formalism of Symmetric Searchable Encryption	40
3.1.2	Correctness	41
3.1.3	Confidentiality	42
3.1.4	Soundness	47
<b>3.2</b>	<b>Leakage in Searchable Encryption</b>	<b>48</b>
3.2.1	An Order Relation over Leakage Functions	48
3.2.2	Commonly Encountered Leakage	48
3.2.3	Efficiency Implies Leakage	51
<b>3.3</b>	<b>The Locality of Searchable Encryption</b>	<b>56</b>
3.3.1	Locality, Read Efficiency and Overlapping Reads	57
3.3.2	A Lower Bound on the Locality of Searchable Encryption	58
3.3.3	Constructions with Improved Locality	58
<b>3.4</b>	<b>Leakage Abuse Attacks</b>	<b>59</b>
3.4.1	Attacks Based on Keyword Frequency	59
3.4.2	File Injection Attacks	60

---

### 3.1 Definitions

In order to go further, and to start proving security theorems, we have to formalize Searchable Encryption and its security.

#### 3.1.1 Formalism of Symmetric Searchable Encryption

A *database*  $DB$  is defined as:

$$DB = \{(\text{ind}_i, W_i) : 1 \leq i \leq D\},$$

with  $\text{ind}_i \in \{0, 1\}^\ell$ ,  $W_i \in \{\{0, 1\}^*\}^*$  and where  $\text{ind}_i$  are distinct *document indices*, represented by  $\ell$ -bit strings, and  $W_i$  is a finite set of *keywords* matching document  $\text{ind}_i$ , represented by binary strings of arbitrary finite length. Note that, in this manuscript, documents are identified with their indices: indeed we focus on index-based searchable encryption, where the documents are encrypted (and stored) separately from the data used to search among these. In addition, let us define:

$$\begin{aligned} D &= |DB| \text{ the number of documents;} \\ W &= \bigcup_{i=1}^D W_i \text{ the set of keywords;} \\ K &= |W| \text{ the number of keywords;} \\ N &= \sum_{i=1}^D |W_i| \text{ the number of document/keyword pairs.} \end{aligned}$$

Let  $DB(w)$  denote the set of documents containing keyword  $w$ :

$$DB(w) = \{\text{ind}_i | w \in W_i \text{ and } (\text{ind}_i, W_i) \in DB\}.$$

The value  $n_w$  is the number of documents matching  $w$ :  $n_w = |DB(w)|$ . Also, we will use the notation  $a_w$  for the total number of entries added to  $w$  (*i.e.* including the entries which were later deleted). We refer to Section 3.2.2 for the formal definition of  $a_w$ .

A *dynamic symmetric searchable encryption (SSE) scheme* is a triple  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$  consisting of one algorithm and two protocols between a client and a server:

- $\text{Setup}(DB)$  is a probabilistic algorithm that takes as input the initial database  $DB$ . It outputs a triple  $(\text{EDB}, K_\Sigma, \sigma)$ , where  $K_\Sigma$  is the master secret key,  $\text{EDB}$  is an encrypted database, and  $\sigma$  is the client's state.
- $\text{Search}(K_\Sigma, q, \sigma; \text{EDB}) = (\text{Search}_C(K_\Sigma, q, \sigma), \text{Search}_S(\text{EDB}))$  is a protocol between the client with input the master secret key  $K_\Sigma$ , the client's internal state  $\sigma$ , and a search query  $q$ ; and the server with input the encrypted database  $\text{EDB}$ .

After completing the  $\text{Search}$  protocol, the client outputs a list  $R$  of results and a new internal state  $\sigma'$ . Both  $R$  and  $\sigma'$  can take the special value  $\perp$  to signify an error or a failure in the execution of the protocol. The server possibly outputs an updated encrypted database  $\text{EDB}'$ .

The query  $q$  can be of any kind: although we will mainly focus on search queries restricted to a single-keyword  $w$  (and hence often identify  $q$  with  $w$ ), this formalism also supports range queries or boolean queries — a search query consisting of a boolean formula  $\phi$  and a set of keywords  $(w_1, \dots, w_n)$  matching the set of documents  $DB(\phi, w_1, \dots, w_n)$  such that

$$\begin{aligned} DB(\phi, w_1, \dots, w_n) &= \{\text{ind}_i | \phi(b_1, \dots, b_n) = \text{true}, \text{ where } b_j = (w_j \in W_i), \\ &\text{and } (\text{ind}_i, W_i) \in DB\}. \end{aligned}$$

More generally,  $DB(q)$  denotes the set of documents matching the query  $q$ .

- $\text{Update}(K_\Sigma, \sigma, \text{op}, \text{in}; \text{EDB}) = (\text{Update}_C(K_\Sigma, \sigma, \text{op}, \text{in}), \text{Update}_S(\text{EDB}))$  is a protocol between the client with input the key  $K_\Sigma$  and internal state  $\sigma$ , an operation  $\text{op}$ , and an input  $\text{in}$  for the operation; and the server with input  $\text{EDB}$ . Again, this formalism covers a wide range of different update operations, such as merges, duplications, *etc.* Yet, this thesis focuses on simpler operations: the update operations are taken from the set  $\{\text{add}, \text{del}\}$ , meaning, respectively, the addition and the deletion of a set of keywords to a document. The input  $\text{in}$  is thus parsed as an index  $\text{ind}$ , pointing to the modified document, and a set  $W$  of keywords to insert or delete. Insertion of a new document is modeled by using a completely new and previously unused index  $\text{ind}$ .

At the end of the execution of the protocol, the client outputs a new state  $\sigma'$ , which can take the special value  $\perp$ , and the server outputs a new encrypted database  $\text{EDB}'$ .

These searchable encryption schemes are called *symmetric* because the same key  $K_\Sigma$  is used for both the updates and the search queries. One could extend the definition to support two different keys: a private key for search and a public key for updates, so that anyone can enrich the encrypted database (*e.g.* by sending an email encrypted with the public key). This setting, called ‘Public key encryption with keyword search’ (PEKS) has been well defined [BDOP04; BKOS07], but in this thesis, we will focus on the symmetric setting. Similarly, some works looked at how to let anyone with a public key issue some search queries [BBO07].

In this formalism, we explicitly separate the client’s state and the key. Informally, we want the key to be fixed while the state is mutable. We do not require a scheme to have a state (beyond the key), and when it is empty, we may omit it from the protocol’s signature. Also, we do not require the size of the state to be upper bounded, *e.g.* by the number of keywords. For example, we could imagine a scheme working only on the client side, with no storage on the server. This would be a perfectly valid (and secure), but very costly scheme. Hence, the size of the client’s state is an important parameter regarding the tradeoff between security and performance.

An important restriction of this definition is *static symmetric searchable encryption*. Such schemes do not support update requests, and hence do not implement the Update protocol. In that case, once the encrypted database has been set up, it is immutable.

### 3.1.2 Correctness

The correctness of an SSE scheme is the basic property we want to ensure: the search protocol must return the correct result for every query, except with negligible probability. We formally define correctness with the security game  $\text{SSECORR}$  described in Figure 3.1.

The game uses the function  $\text{Apply}$  that outputs  $\text{DB}$  updated according to the input operation  $\text{op}$ , and the input  $\text{in}$  for that operation. Also, note that, accordingly to the protocol formalism described in Section 3.1.1, in the Search and the Update game procedures,  $\tau$  is the transcript of the client – *i.e.* the messages sent by the client to the server – and that we omit the transcript of the server.

In this game, the adversary tries to construct a sequence of operations leading to an incorrectly answered search query, while respecting the protocols. In particular, he is not allowed to modify the encrypted database, nor the messages (the transcripts) between the client and the server.

**Definition 3.1** (SSE Correctness). *Let  $\Sigma$  be an SSE scheme. For an adversary  $A$ , the advantage  $\text{Adv}_{\Sigma, A}^{\text{SSE-corr}}(\lambda)$  of  $A$  in the correctness game is defined as*

$$\text{Adv}_{\Sigma, A}^{\text{SSE-corr}}(\lambda) = \mathbb{P}[\text{SSECORR}_\Sigma^A(\lambda) = 1].$$

*An SSE scheme  $\Sigma$  is correct if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\Sigma, A}^{\text{SSE-corr}}(\lambda)$  is negligible in  $\lambda$ .*

<pre> Init(DB)   (EDB, <math>K_\Sigma, \sigma</math>) <math>\stackrel{\\$}{\leftarrow}</math> Setup(DB)   <b>return</b> EDB Search(<math>q</math>)   (<math>R, \sigma, \tau</math>; EDB) <math>\stackrel{\\$}{\leftarrow}</math> Search(<math>K_\Sigma, \sigma, q</math>; EDB)   <b>if</b> <math>R \neq \text{DB}(q)</math> <b>or</b> <math>\sigma = \perp</math>     <b>win</b> <math>\leftarrow</math> true   <b>return</b> <math>\tau</math> Final()   <b>return</b> win </pre>	<pre> Update(op, in)   (<math>\sigma', \tau</math>; EDB') <math>\stackrel{\\$}{\leftarrow}</math> Update(<math>K_\Sigma, \sigma, \text{op}, \text{in}</math>; EDB)   <b>if</b> <math>\sigma' = \perp</math> <b>then</b>     <b>win</b> <math>\leftarrow</math> true   <b>else</b>     DB <math>\leftarrow</math> Apply(DB, op, in)     EDB <math>\leftarrow</math> EDB', <math>\sigma \leftarrow \sigma'</math>   <b>end if</b>   <b>return</b> <math>\tau</math> </pre>
--	--

**Figure 3.1** – SSECORR $_\Sigma$ : Correctness game for the SSE scheme  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ .

### 3.1.3 Confidentiality

We saw in Section 1.2.1 that it looks hard to construct a scheme that is both secure and efficient, security meaning here confidentiality of the query *and* of the database. To formalize this assertion, we first have to give a rigorous definition of the confidentiality of an SSE scheme, in particular if we also want to have fine-grained definitions for not perfectly confidential constructions.

It is interesting to notice that the Song *et al.* paper [SWP00] does not give any formal security definition, although the authors already underline that the server could learn a lot of information using statistical techniques if the client user asks many queries. Indeed, the authors only show that their construction is a secure encryption scheme, but do not consider the information leaked by the queries.

The first formal security definition specifically designed for searchable encryption applications comes from Goh [Goh03], who gave a definition of secure indexes – which can not only be used for searchable encryption, but more generally for any application requiring fast access to an information using a trapdoor. Unfortunately, the IND-CKA notion defined by Goh does not protect the confidentiality of trapdoors, but only of the index.

Chang and Mitzenmacher [CM05] tried to integrate the confidentiality of the trapdoors in their simulation-based security definition. Unfortunately, Curtmola *et al.* showed in [CGKO06] that this definition was flawed (a problem with the order of quantifiers). Also, it is non-adaptive and might not reflect the behavior of an adversarial server.

Modern definitions to formalize the secrecy of an SSE scheme come from [CGKO06]. In this paper, the authors give two definitions, one based on indistinguishability, the other based on simulatability, both using the notion of leakage (a.k.a *trace* in the paper). Indeed, the leakage is formally taken into account and plays an important role in the security definitions: the indistinguishability-based definition states that two executions of SSE protocols with the same leakage are indistinguishable, while the simulation-based definition states that an execution of the SSE protocol can be simulated using the leakage. Informally, both definitions ensure that the server should not learn any information beyond the leakage (which is a parameter of the definitions).

#### 3.1.3.1 Leakage Function

Before going further, it is essential to clearly formalize this leakage. First let us define a history.

**Definition 3.2** (Database and queries history). *An history  $H$  is a tuple  $H = (\text{DB}, r_1, \dots, r_m)$*

consisting of a database  $DB$  and  $m$  queries  $r_1, \dots, r_m$ . Each query  $r_i$  can either be a search query  $r_i = q_i$ , or an update query  $r_i = (\text{op}_i, \text{in}_i)$ .

To do so, as explained before, the definition will be parametrized using a *leakage function*  $\mathcal{L}$ , more exactly a triple of stateful algorithms  $(\mathcal{L}^{\text{Stp}}, \mathcal{L}^{\text{Srch}}, \mathcal{L}^{\text{Updt}})$ , capturing what is leaked by, respectively, the setup algorithm, the search protocol and the update protocol.

This notation, introduced by Chase and Kamara for the generic case of structured encryption in [CK10], generalizes the trace definition of Curtmola *et al.* [CGKO06]. Because the leakage function is stateful, it will not be necessary to pass the whole history as an argument of the leakage function every time, as in [CJJ+13].

In the following, we will slightly overload the notations, and use  $\mathcal{L}(H)$  for an history  $H = (DB, r_1, \dots, r_m)$  to denote  $(\mathcal{L}^{\text{Stp}}(DB), \mathcal{L}(r_1), \dots, \mathcal{L}(r_m))$  where  $\mathcal{L}(r_i) = \mathcal{L}^{\text{Srch}}(q_i)$  if  $r_i$  is a search query, and  $\mathcal{L}(r_i) = \mathcal{L}^{\text{Updt}}(\text{op}_i, \text{in}_i)$  if  $r_i$  is an update query.

### 3.1.3.2 Honest-but-curious Adversaries

We start with the simpler setting of an ‘honest-but-curious’ server, *i.e.* a server who tries to learn as much information as he could, without deviating from the protocols. In the next section, we will see how to generalize these definitions to a ‘malicious’ server who can deviate from the prescribed behavior.

**Indistinguishability-based definition.** The indistinguishability-based security definition of Curtmola *et al.* [CGKO06] can be reformulated (equivalently) using a security game,  $\text{SSEIND}_{\Sigma, \mathcal{L}}$ , parametrized by the scheme  $\Sigma$  and the leakage function  $\mathcal{L}$ , defined in Figure 3.2.

The  $\text{SSEIND}$  game picks a random bit  $b$  and the adversary’s goal is to guess  $b$ . To do so, he submits two histories  $H_0$  and  $H_1$ , and receives the encrypted database and the transcript  $\tau$  of the queries corresponding to  $H_b$ . However, the histories submitted by the adversary must satisfy a very important constraint: the leakage must be the same for both histories. Otherwise, the game aborts as soon as the adversary submits two queries (or two databases) having a different leakage. Finally, the adversary outputs a bit  $b'$ , and wins the game if he successfully guesses  $b$ .

<pre> Init(<math>DB_0, DB_1</math>)   if <math>\mathcal{L}^{\text{Stp}}(DB_0) \neq \mathcal{L}^{\text{Stp}}(DB_1)</math>     Abort game   <math>b \xleftarrow{\\$} \{0, 1\}</math>   <math>(EDB, K_{\Sigma}, \sigma) \xleftarrow{\\$} \text{Setup}(DB_b)</math>   return EDB Final(<math>b'</math>)   return <math>b = b'</math> </pre>	<pre> Search(<math>q_0, q_1</math>)   if <math>\mathcal{L}^{\text{Srch}}(q_0) \neq \mathcal{L}^{\text{Srch}}(q_1)</math>     Abort game   <math>(R, \sigma, \tau; EDB) \xleftarrow{\\$} \text{Search}(K_{\Sigma}, \sigma, q_b; EDB)</math>   return <math>\tau</math> Update(<math>(\text{op}_0, \text{in}_0), (\text{op}_1, \text{in}_1)</math>)   if <math>\mathcal{L}^{\text{Updt}}(\text{op}_0, \text{in}_0) \neq \mathcal{L}^{\text{Updt}}(\text{op}_1, \text{in}_1)</math>     Abort game   <math>(\sigma, \tau; EDB) \xleftarrow{\\$} \text{Update}(K_{\Sigma}, \sigma, \text{op}_b, \text{in}_b; EDB)</math>   return <math>\tau</math> </pre>
---	--

**Figure 3.2** –  $\text{SSEIND}_{\Sigma, \mathcal{L}}$ : Indistinguishability game for the SSE scheme  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ , with the leakage function  $\mathcal{L} = (\mathcal{L}^{\text{Stp}}, \mathcal{L}^{\text{Srch}}, \mathcal{L}^{\text{Updt}})$ .

**Definition 3.3** (Indistinguishability-based confidentiality of SSE). *Let  $\Sigma$  be an SSE scheme. For an adversary  $A$ , the advantage  $\text{Adv}_{\Sigma, \mathcal{L}, A}^{\text{SSE-ind}}(\lambda)$  of  $A$  in the indistinguishability-based confidentiality game is defined as*

$$\text{Adv}_{\Sigma, \mathcal{L}, A}^{\text{SSE-ind}}(\lambda) = \left| \frac{1}{2} - \mathbb{P}[\text{SSEIND}_{\Sigma, \mathcal{L}}^A(1^\lambda) = 1] \right|.$$

An SSE scheme  $\Sigma$  is  $\mathcal{L}$ -adaptively-indistinguishability secure if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\Sigma, \mathcal{L}, A}^{\text{SSE-ind}}(\lambda)$  is negligible in  $\lambda$ .

For this definition to make sense, as mentioned in [CGKO06], we have to make sure that every polynomial-time constructible history  $H_0$  has a corresponding polynomial-time constructible history  $H_1$  such that  $\mathcal{L}(H_0) = \mathcal{L}(H_1)$ . Such histories are called *non-singular*. In the following, unless mentioned otherwise, we suppose that the histories are non-singular.

We can also extend this definition to the case where  $\mathcal{L}$  is not a deterministic leakage function. Indeed, we will see that some information leaked by the scheme can depend on some randomness, for example when using probabilistic padding. In that, we cannot require that two histories have the same leakage, but instead that the leakage distribution is indistinguishable between the two histories, i.e. that  $\mathcal{L}^{\text{Stp}}(\text{DB}_0) \approx \mathcal{L}^{\text{Stp}}(\text{DB}_1)$  at setup,  $\mathcal{L}^{\text{Srch}}(q_0) \approx \mathcal{L}^{\text{Srch}}(q_1)$  for a search query and  $\mathcal{L}^{\text{Updt}}(\text{op}_0, \text{in}_0) \approx \mathcal{L}^{\text{Updt}}(\text{op}_1, \text{in}_1)$  for an update query.

We could also have extended the definition to the case where the leakage of two histories are computationally indistinguishable, but this would complicate this definition a lot, and in this thesis, we will not use that type of leakage function.

**Simulation-based definition.** The simulation-based security definition states that an execution of the SSE protocols, more exactly its transcripts, is computationally indistinguishable from a simulated execution which only uses the leakage, and, in particular, does not use the secret key.

For this definition, we use the real-world versus ideal-world paradigm, as in the MPC literature [Gol04]. Two games are constructed,  $\text{SSEReal}$  and  $\text{SSEIdeal}$ . In the real game the adversary chooses a database  $\text{DB}$  and gets back  $\text{EDB}$ . Then, it adaptively runs Search and Update protocols on inputs of its choice, and is given the transcripts of these protocols. The Final call simply forwards the bit  $b$  output by the adversary. In the ideal game, these transcripts are generated by a simulator  $S$ , an efficient algorithm, helped by the outputs of the leakage function. Finally, the scheme will be secure if no efficient adversary can distinguish between the real and the ideal games. Both games are described in Figure 3.3.

**Definition 3.4** (Simulation-based confidentiality of SSE). *Let  $\Sigma$  be an SSE scheme. For an adversary  $A$  and a simulator  $S$ , the advantage  $\text{Adv}_{\Sigma, S, \mathcal{L}, A}^{\text{SSE-sim}}(\lambda)$  of  $A$  in the indistinguishability-based confidentiality game is defined as*

$$\text{Adv}_{\Sigma, S, \mathcal{L}, A}^{\text{SSE-sim}}(\lambda) = \left| \mathbb{P}[\text{SSEReal}_{\Sigma}^A(1^\lambda) = 1] - \mathbb{P}[\text{SSEIdeal}_{\Sigma, S, \mathcal{L}}^A(1^\lambda) = 1] \right|.$$

An SSE scheme  $\Sigma$  is said to be  $\mathcal{L}$ -adaptively-semantically secure if for any polynomial-time adversary  $A$ , there exists a polynomial-time simulator  $S$  such that  $\text{Adv}_{\Sigma, S, \mathcal{L}, A}^{\text{SSE-sim}}(\lambda)$  is negligible in  $\lambda$ .

The intuition behind this definition, is that, because the real world is indistinguishable from the ideal world, an adversary running an  $\mathcal{L}$ -adaptively-semantically secure SSE scheme  $\Sigma$  cannot learn more information than what is leaked.

In most SSE simulation-based confidentiality proofs of this thesis, and more generally in the searchable encryption literature, the simulator will not depend on the adversary  $A$  (the simulator is



<p><u>SSEReal<math>_{\Sigma}</math></u>  <u>Init(DB)</u>  <math>(\text{EDB}, K_{\Sigma}, \sigma) \xleftarrow{\\$} \text{Setup}(\text{DB})</math>  <b>return</b> EDB</p> <p><u>Search(<math>q</math>)</u>  <math>(R, \sigma', \tau; \text{EDB}) \xleftarrow{\\$} \text{Search}(K_{\Sigma}, \sigma, q; \text{EDB})</math>  <b>if</b> <math>R \neq \perp</math>  <math>\sigma \leftarrow \sigma'</math>  <b>return</b> <math>\tau</math></p> <p><u>Update(op, in)</u>  <math>(\sigma', \tau; \text{EDB}') \xleftarrow{\\$} \text{Update}(K_{\Sigma}, \sigma, \text{op}, \text{in}; \text{EDB})</math>  <b>if</b> <math>\sigma' \neq \perp</math>  <math>\sigma \leftarrow \sigma', \text{EDB} \leftarrow \text{EDB}'</math>  <b>return</b> <math>\tau</math></p>	<p><u>SSEIdeal<math>_{\Sigma, S, \mathcal{L}}</math></u>  <u>Init(DB)</u>  <math>\text{EDB} \xleftarrow{\\$} S(\mathcal{L}^{\text{Stp}}(\text{DB}))</math>  <b>return</b> EDB</p> <p><u>Search(<math>q</math>)</u>  <math>\tau \xleftarrow{\\$} S(\mathcal{L}^{\text{Srch}}(q))</math>  <b>return</b> <math>\tau</math></p> <p><u>Update(op, in)</u>  <math>\tau \xleftarrow{\\$} S(\mathcal{L}^{\text{Updt}}(\text{op}, \text{in}))</math>  <b>return</b> <math>\tau</math></p>
---	--

**Figure 3.3** – SSE security games SSEReal (left) and SSEIdeal (right) for honest-but-curious adversaries.

*universal*): we could permute the quantifiers on  $A$  and  $S$  in the security definition. In particular, we will not use complex proofs tools such as adversary rewinding. However, the definition, as written above, does not prevent this.

**Relations between indistinguishability-based and simulation-based definitions.** It has been shown in [CGKO06], that, in the non-adaptive setting, the two security definitions are equivalent (with a tight reduction). However, this is not true against an adaptive adversary, although simulation-based security implies indistinguishability-based security in general.

In particular, the separation is strict when we are not in the random oracle model, and when the search tokens are *succinct* (their size does not depend on the size of the database, on the number of documents, nor on the number of distinct keywords, only on the security parameter). Chase and Kamara show in [CK10, Section 4, Remark 1] that a SSE scheme that is simulation-based-secure in the standard model must have search tokens essentially as large as the number of documents. As we will see in the following, this limitation does not hold in the ROM, nor does it hold in the standard model with an indistinguishability-based definition. Actually, some existing constructions (e.g. the ones in [CGKO06; CJJ+13]) are adaptively secure in the ROM following Definition 3.4, and can be easily shown secure against adaptive adversary in the standard model, according to Definition 3.3. These schemes have succinct tokens (their size only depend on the security parameter), and hence cannot be simulation-secure in the standard model, showing a strict separation between the SSE confidentiality security definitions in the standard model.

### 3.1.3.3 Malicious Adversaries

Unfortunately, the previous games do not capture an adversary who tries to get additional information by deviating from the protocols. This is particularly important for schemes whose protocols use more than a single round-trip, e.g. ORAM-inspired constructions such as in [SPS14; GMP16]. A

<p><u>Init(DB<sub>0</sub>, DB<sub>1</sub>)</u>  <b>if</b> <math>\mathcal{L}^{\text{Stp}}(\text{DB}_0) \neq \mathcal{L}^{\text{Stp}}(\text{DB}_1)</math>            Abort game  <math>b \xleftarrow{\\$} \{0, 1\}</math>  <math>(\text{EDB}, K_\Sigma, \sigma) \xleftarrow{\\$} \text{Setup}(\text{DB}_b)</math>  <b>return</b> EDB</p> <p><u>Final(b')</u>  <b>return</b> <math>b = b'</math></p>	<p><u>Search(q<sub>0</sub>, q<sub>1</sub>)</u>  <b>if</b> <math>\mathcal{L}^{\text{Srch}}(q_0) \neq \mathcal{L}^{\text{Srch}}(q_1)</math>            Abort game  <math>(R, \sigma, \tau; \text{EDB}) \xleftarrow{\\$} \text{Search}_C(K_\Sigma, \sigma, q_b; \text{EDB}) \leftrightarrow A</math>  <b>return</b> <math>\tau</math></p> <p><u>Update((op<sub>0</sub>, in<sub>0</sub>), (op<sub>1</sub>, in<sub>1</sub>))</u>  <b>if</b> <math>\mathcal{L}^{\text{Updt}}(\text{op}_0, \text{in}_0) \neq \mathcal{L}^{\text{Updt}}(\text{op}_1, \text{in}_1)</math>            Abort game  <math>(\sigma, \tau; \text{EDB}) \xleftarrow{\\$} \text{Update}_C(K_\Sigma, \sigma, \text{op}_b, \text{in}_b; \text{EDB}) \leftrightarrow A</math>  <b>return</b> <math>\tau</math></p>
---	---

**Figure 3.4** – Generalized security game SSEIND capturing malicious adversaries. The notation  $\leftrightarrow A$  represents interactions with the adversary.

single round-trip scheme secure against honest adversaries is secure against malicious ones because no adversary can, by construction, influence the client's transcript by providing false or incomplete responses. To the contrary, when multiple round-trips are involved, the adversary can trick the client by sending an incorrect message so that the client reveals sensitive information later in the search or update protocol.

Fortunately, it is quite easy to adapt the security games SSEIND, SSEReAL and SSEIDEAL to capture such adversaries. To do so, instead of running both sides of the Search and Update protocols, the game will run the client-side part of the protocol and let the adversary interact with him for the server-side part. These modifications are described in Figure 3.4 (for the SSEIND game) and Figure 3.5 (for the SSEReAL and SSEIDEAL games).

<p>SSEReAL<sub>Σ</sub>  <u>Init(DB)</u>  <math>(\text{EDB}, K_\Sigma, \sigma) \xleftarrow{\\$} \text{Setup}(\text{DB})</math>  <b>return</b> EDB</p> <p><u>Search(q)</u>  <math>(R, \sigma', \tau; \text{EDB}) \xleftarrow{\\$} \text{Search}_C(K_\Sigma, \sigma, q) \leftrightarrow A</math>  <b>if</b> <math>R \neq \perp</math>  <math>\sigma \leftarrow \sigma'</math>  <b>return</b> <math>\tau</math></p> <p><u>Update(op, in)</u>  <math>(\sigma', \tau; \text{EDB}') \xleftarrow{\\$} \text{Update}_C(K_\Sigma, \sigma, \text{op}, \text{in}) \leftrightarrow A</math>  <b>if</b> <math>\sigma' \neq \perp</math>  <math>\sigma \leftarrow \sigma', \text{EDB} \leftarrow \text{EDB}'</math>  <b>return</b> <math>\tau</math></p>	<p>SSEIDEAL<sub>Σ,S,ℒ</sub>  <u>Init(DB)</u>  <math>\text{EDB} \xleftarrow{\\$} S(\mathcal{L}^{\text{Stp}}(\text{DB}))</math>  <b>return</b> EDB</p> <p><u>Search(q)</u>  <math>\tau \xleftarrow{\\$} S(\mathcal{L}^{\text{Srch}}(q)) \leftrightarrow A</math>  <b>return</b> <math>\tau</math></p> <p><u>Update(op, in)</u>  <math>\tau \xleftarrow{\\$} S(\mathcal{L}^{\text{Updt}}(\text{op}, \text{in})) \leftrightarrow A</math>  <b>return</b> <math>\tau</math></p>
---	--

**Figure 3.5** – SSE security games SSEReAL (left) and SSEIDEAL (right). The notation  $\leftrightarrow A$  represents interactions with the adversary.

Note that, if  $A$  exactly follows  $\text{Search}_S$  and  $\text{Update}_S$ , respectively for the search and the update

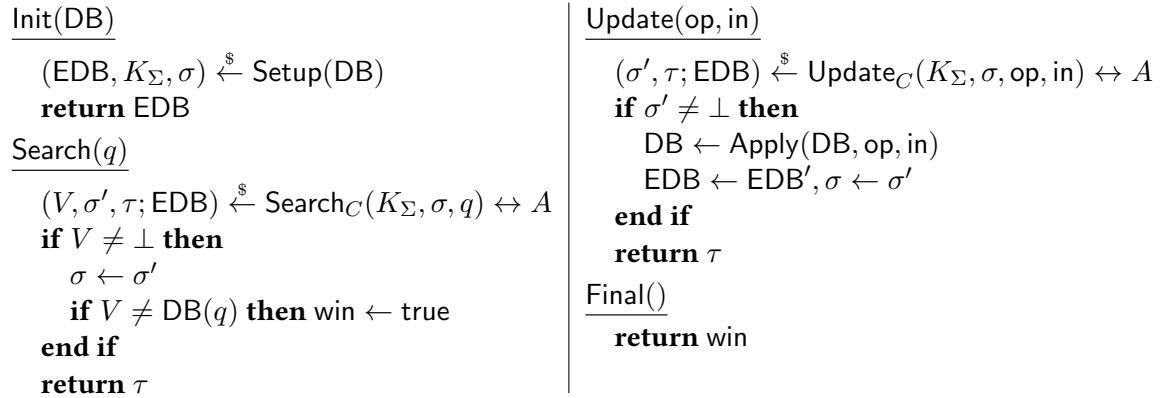
queries, we end up with the same games as in Section 3.1.3.2. Hence, we will use these generalized games in the following.

Finally, once the games have been updated, we do not have to modify the security definitions themselves: for the security against malicious adversaries, both Definition 3.3 and Definition 3.4 remain valid.

### 3.1.4 Soundness

The last important security definition we will use is about the notion of *soundness*. As for regular soundness definitions, *e.g.* for provers, it describes the fact that no adversary can cheat the client, and make him accept incorrect search results.

To give a proper soundness definition, we use the game SSESOUND defined in Figure 3.6. The game closely follows the game used to define soundness of interactive provers [Gol04]: the client must not accept an invalid search result. Also, the dynamism of the database raises a difficult point: the verification has to be done over the current version of the database, and this one must not be modifiable undetectably by a malicious server. Hence, SSESOUND does not apply the update operation on the database when the client rejects the execution of the Update protocol with the server.



**Figure 3.6** – SSESOUND $_\Sigma$ : Soundness game for scheme  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ .

One could try to interpret the soundness game as a malicious server variant of the correctness game defined in Section 3.1.2. However, it is important to see that the correctness definition prevents the Search or the Update protocols to return  $\perp$  when the adversary does not behave maliciously, while the soundness game, in the opposite, ensures that this happens if the adversary tries to cheat. SSECORR and SSESOUND are similar-looking games, but are also crucially different in their security goals.

**Definition 3.5** (SSE Soundness). *Let  $\Sigma$  be an SSE scheme. For an adversary  $A$ , the advantage  $\text{Adv}_{\Sigma, A}^{\text{SSE-snd}}(\lambda)$  of  $A$  in the soundness game is defined as*

$$\text{Adv}_{\Sigma, A}^{\text{SSE-snd}}(\lambda) = \mathbb{P}[\text{SSESOUND}_\Sigma^A(1^\lambda) = 1].$$

*An SSE scheme  $\Sigma$  is sound if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\Sigma, A}^{\text{SSE-snd}}(\lambda)$  is negligible in  $\lambda$ .*

This definition can be seen as a generalization of the reliability definition of Kurosawa and Ohtaki [KO12]: in that paper, the authors study the case of single round-trip constructions (also static ones, but they later extended their definitions to the dynamic setting in [KO13]).

## 3.2 Leakage in Searchable Encryption

Now that we have a formal security definition using the notion of leakage, we can study it more thoroughly. This section explains how we can compare the leakage of different schemes — and gives a formal definition to “leaking less than” — describes a few commonly encountered leakage components, and shows that the absence of leakage implies the inefficiency of the searchable encryption scheme.

### 3.2.1 An Order Relation over Leakage Functions

We want to give a formal definition to the proposition “ $\mathcal{L}_1$  leaks less than  $\mathcal{L}_2$ ”. By that, we mean that  $\mathcal{L}_1$  gives less information about the database and the queries to the simulator than  $\mathcal{L}_2$ , or, said otherwise, that every information given by  $\mathcal{L}_1$  can be inferred from  $\mathcal{L}_2$ . This can be formalized by the fact that  $\mathcal{L}_1$  is a function of  $\mathcal{L}_2$ : a function information-theoretically compresses the information, or returns the same information.

**Definition 3.6** (The order  $\preceq$  on leakage functions). *Let  $\mathcal{L}_1 = (\mathcal{L}_1^{\text{Stp}}, \mathcal{L}_1^{\text{Srch}}, \mathcal{L}_1^{\text{Updt}})$  and  $\mathcal{L}_2 = (\mathcal{L}_2^{\text{Stp}}, \mathcal{L}_2^{\text{Srch}}, \mathcal{L}_2^{\text{Updt}})$  be two leakage functions. We say that  $\mathcal{L}_1$  leaks less than  $\mathcal{L}_2$ , denoted by  $\mathcal{L}_1 \preceq \mathcal{L}_2$  if and only if, there exists a triple of stateful polynomial-time algorithms  $\mathcal{T} = (\mathcal{T}^{\text{Stp}}, \mathcal{T}^{\text{Srch}}, \mathcal{T}^{\text{Updt}})$ , such that, for any database DB and sequence of queries  $(r_1, \dots, r_n)$ ,*

$$\begin{aligned} \mathcal{L}_1^{\text{Stp}}(\text{DB}) &= \mathcal{T}^{\text{Stp}} \circ \mathcal{L}_2^{\text{Stp}}(\text{DB}), \\ \forall 1 \leq i \leq n, \mathcal{L}_1^{\text{Srch}}(r_i) &= \mathcal{T}^{\text{Srch}} \circ \mathcal{L}_2^{\text{Srch}}(r_i) \text{ if } r_i \text{ is a search query,} \\ \text{and } \mathcal{L}_1^{\text{Updt}}(r_i) &= \mathcal{T}^{\text{Updt}} \circ \mathcal{L}_2^{\text{Updt}}(r_i) \text{ if } r_i \text{ is an update query.} \end{aligned}$$

This is also denoted by  $\mathcal{L}^1 = \mathcal{T} \circ \mathcal{L}^2$ .

In other words,  $\mathcal{L}_1$ 's output is *simulatable* from  $\mathcal{L}_2$ 's output. It is clear that  $\preceq$  is reflexive and transitive, but not anti-symmetric, and thus is a preorder. We can define the equivalence relation associated with this preorder :

**Definition 3.7.** *The leakage function  $\mathcal{L}_1$  is equivalent to  $\mathcal{L}_2$  (denoted  $\mathcal{L}_1 \simeq \mathcal{L}_2$ ) if and only if  $\mathcal{L}_1 \preceq \mathcal{L}_2$  and  $\mathcal{L}_2 \preceq \mathcal{L}_1$*

A nice application of this relation is given by the following claim : we only have to prove security for the *smallest* leakage function.

**Proposition 3.1.** *Let  $\mathcal{L}_1$  and  $\mathcal{L}_2$  be two leakage functions such that  $\mathcal{L}_1 \preceq \mathcal{L}_2$ . If  $\Sigma$  is a  $\mathcal{L}_1$ -adaptively-secure SSE scheme then  $\Sigma$  is  $\mathcal{L}_2$ -adaptively-secure.*

*Proof.* Let  $A$  be an adversary, and  $\mathcal{T}$  such that  $\mathcal{L}_1 = \mathcal{T} \circ \mathcal{L}_2$ . As  $\Sigma$  is  $\mathcal{L}_1$ -adaptively-secure, there exists  $S$  such that  $\text{Adv}_{\Sigma, S, \mathcal{L}_1, A}^{\text{SSE-sim}}(\lambda) \leq \text{negl}(\lambda)$ . If  $S' = S \circ \mathcal{T}$ , i.e.  $S'^{\text{Stp}} = S^{\text{Stp}} \circ \mathcal{T}^{\text{Stp}}$  for the setup,  $S'^{\text{Srch}} = S^{\text{Srch}} \circ \mathcal{T}^{\text{Srch}}$  for a search query, and  $S'^{\text{Updt}} = S^{\text{Updt}} \circ \mathcal{T}^{\text{Updt}}$  for an update query, then

$$\text{Adv}_{\Sigma, S', \mathcal{L}_2, A}^{\text{SSE-sim}}(\lambda) = \text{Adv}_{\Sigma, S \circ \mathcal{T}, \mathcal{L}_2, A}^{\text{SSE-sim}}(\lambda) = \text{Adv}_{\Sigma, S, \mathcal{T} \circ \mathcal{L}_2, A}^{\text{SSE-sim}}(\lambda) = \text{Adv}_{\Sigma, S, \mathcal{L}_1, A}^{\text{SSE-sim}}(\lambda) \leq \text{negl}(\lambda).$$

□

### 3.2.2 Commonly Encountered Leakage

The leakage of searchable encryption often consists of similar elements, which we will describe in this section.

**Response hiding and response revealing schemes.** First of all, schemes may leak the result set itself. This leakage can be caused by the scheme itself (when the results of the search are transmitted in the clear to the client), or by the fact that the client, upon the receipt of the results, accesses the matching documents stored on the server, without relying on an access-pattern-hiding protocol, *e.g.* ORAM.

In the literature, the two settings are however clearly separated: schemes explicitly revealing the results are called *response revealing*. In the opposite, schemes whose results are kept encrypted when sent to the client are called *response hiding*.

Also, note that a response revealing scheme, because it leaks the results, also leaks very useful information for a statistical attack against the queries, such as the number of co-occurrences (*cf.* the next section).

**Size of the database, number of distinct keywords, result count ...** The server will always see the size of the encrypted database, as he stores it. Hence, unless the database is padded by a massive number of fake entries, unrelated with the number of real entries, he will learn the number of document/keyword pairs  $N$  in the dataset.

Similarly, one element that is leaked by all efficient construction is the number of results of a search query. Unless the result set is padded, the server will learn the number of matches of a query, and will be able to use this information to retrieve the queried keyword (*cf.* next section).

One less common leakage is the number of unique keywords in the database. Some schemes (often dynamic ones) store an array of keyword-indexed elements on the server, to avoid storing in on the client (*e.g.* [CJJ+14] or the schemes presented in Sections 4.5, 5.4, 4.6, and 6.3), leaking by the same the number of different keywords of the database. To this date, this leakage is not considered harmful, as no attack makes use of it.

**Repetition of queries.** Another type of information that the schemes not based on generic tools (FHE, ORAM, ...) leak is the repetition of some queries. Indeed, in many schemes, some of the tokens sent by the client to the server, during either the search or the update protocol, are deterministically generated, *e.g.* with a PRF. As a consequence, if a token sent to the server during a search query is generated using only the queried keyword, he will immediately see if this query is repeated. Similarly, a construction that does not hide the access pattern of updates will often leak the repetition of updated keywords.

This leakage can be formally defined as follows. The leakage function  $\mathcal{L}$  will keep in its state the *query list*  $Q$ : the list of all queries issued so far, and whose entries are  $(i, w)$  for a search query on keyword  $w$ , or  $(i, \text{op}, \text{in})$  for an  $\text{op}$  update query with input  $\text{in}$ . The integer  $i$  is a timestamp, initially set to 0, and is incremented at each query. The *search, update and query patterns*, respectively denoted  $\text{sp}(x)$ ,  $\text{up}(x)$ , and  $\text{qp}(x)$ , are then defined as

$$\begin{aligned} \text{sp}(x) &= \{j \mid (j, x) \in Q\} \text{ (only matches search queries)} \\ \text{up}(x) &= \{j \mid (j, \text{op}, \text{in}) \in Q \text{ and } x \text{ refers to in}\} \text{ (only matches update queries)} \\ \text{qp}(x) &= \{j \mid (j, x) \in Q \text{ or } (j, \text{op}, \text{in}) \in Q \text{ and } x \text{ refers to in}\} \text{ (matches both)}. \end{aligned}$$

The search pattern of the keyword  $w$  corresponds to the list of search queries' timestamps whose searched keyword was  $w$ . The update pattern of  $w$  is the list of timestamps of queries corresponding to the update on  $w$ . The query pattern of keyword  $w$  is the list of queries involving  $w$ , both for search and update.

**Keyword's history.** One last important element when studying the leakage of (dynamic) searchable encryption schemes is what we call the *history* of a keyword. Many dynamic schemes leak the point in time when a document matching a keyword was inserted, without leaking the query or the update pattern. We need a way to formalize such leakage, and it is the point of the keyword's history (not to be confused with the database and query history, introduced in Section 3.1.3.1).

The function  $\text{Hist}(w)$  outputs the list of all updates on keyword  $w$ : each element of the list is a tuple  $(u, \text{op}, \text{ind})$  where  $\text{ind}$  is the updated index,  $\text{op}$  the operation, and  $u$  the timestamp of the update. For completeness, we prepend  $\text{Hist}(w)$  with the set  $\text{DB}_0(w)$  of documents matching  $w$  in the initial database, the database used as input of the Setup algorithm. As an example, if there are three documents  $D_{\text{ind}_1}$ ,  $D_{\text{ind}_2}$  and  $D_{\text{ind}_3}$  matching  $w$ , such that  $D_{\text{ind}_1}$  was in the initial dataset,  $D_{\text{ind}_2}$  was inserted at update 3,  $D_{\text{ind}_3}$  at update 7, and then  $D_{\text{ind}_2}$  was deleted at update 42,  $\text{Hist}(w)$  will be  $[(\text{ind}_1), (4, \text{add}, \text{ind}_2), (7, \text{add}, \text{ind}_3), (42, \text{del}, \text{ind}_2)]$ . Formally,  $\text{Hist}(w)$  can be defined as

$$\text{Hist}(w) = [\text{DB}_0(w), \{(j, \text{op}, \text{ind}) \mid (j, \text{op}, \text{in}) \in Q \text{ and } (w, \text{ind}) \text{ appears in in}\}].$$

We also denote  $\text{AddDB}(w)$  the list of documents historically added to DB matching keyword  $w$ , in the order of insertion. In particular, it includes documents that have been added and later deleted, or even documents that have been added twice:

$$\text{AddDB}(w) = [\text{ind} \mid (u, \text{add}, (w, \text{ind})) \in Q], \text{ ordered according to } u.$$

We will often have schemes whose search complexity on  $w$  is linear in  $a_w$ , the number of inserted entries whose keyword is  $w$ :

$$a_w = |\text{AddDB}(w)| = |\{(u, \text{add}, (w, \text{ind})) \in Q\}|.$$

In Chapter 5, we will also use two more leakage components. We defer the explanation of why these are important to that chapter, but mention them here already, for completeness. The first, the timestamped result set  $\text{TimeDB}(w)$  of  $w$  is the list of all documents matching  $w$ , excluding the deleted ones, together with the timestamp of when they were inserted in the database. Formally,  $\text{TimeDB}(w)$  can be constructed from the query list  $Q$  as follows:

$$\text{TimeDB}(w) = \{(u, \text{ind}) \mid (u, \text{add}, (w, \text{ind})) \in Q \text{ and } \forall u' > u, (u', \text{del}, (w, \text{ind})) \notin Q\}.$$

In particular  $\text{DB}(w)$  can be reinterpreted from  $\text{TimeDB}(w)$ :

$$\text{DB}(w) = \{\text{ind} \mid \exists u \text{ s.t. } (u, \text{ind}) \in \text{TimeDB}(w)\} \cup \text{DB}_0(w).$$

Note that  $\text{TimeDB}$  is completely oblivious to any document added to  $\text{DB}(w)$  and later removed, including any related addition or deletion update and their timestamp; but retains all other information.

Finally, the *deletion history*  $\text{DelHist}(w)$  of  $w$  is the list of timestamps for all deletion operations on keyword  $w$ , together with the insertion timestamp of the entry each deletion removes. Formally,  $\text{DelHist}(w)$  is constructed as:

$$\text{DelHist}(w) = \{(u^{\text{add}}, u^{\text{del}}) \mid \exists \text{ind s.t. } (u^{\text{del}}, \text{del}, (w, \text{ind})) \in Q \text{ and } (u^{\text{add}}, \text{add}, (w, \text{ind})) \in Q\}.$$

### 3.2.3 Efficiency Implies Leakage

One crucially important question about searchable encryption is the one of the tradeoff between the security of constructions and their performance. We would like to design efficient schemes with the best possible security, or, symmetrically, design highly secure schemes with the best performance.

In the next section, we will see that the leakage can be very efficiently used to break the confidentiality of queries or of the database. In particular, legacy-compatible constructions are not very secure, although being very efficient (asymptotically as efficient as unencrypted databases). In this section, we will see that some minimal leakage is absolutely necessary to achieve even reasonable performance.

In almost every construction, except for the ones based on ORAM, the search pattern is leaked: the server learns the repetition of search queries. One can wonder if the cost of ORAM is necessary to hide the search pattern.

We can actually show a result (Theorem 3.2) similar to the ORAM lower bound of Goldreich and Ostrovsky [GO96], adapted to the setting of searchable encryption. Before stating this result, we introduce a few notations. Let  $\mathcal{L}_{\overline{\text{sp}}}$  be the leakage function (for static schemes) such that

$$\begin{aligned}\mathcal{L}_{\overline{\text{sp}}}^{\text{Stp}}(\text{DB}) &= (N, K), \\ \mathcal{L}_{\overline{\text{sp}}}^{\text{SrcH}}(w) &= n_w.\end{aligned}$$

To do so, we could write a simple reduction from ORAM to SSE (*i.e.* implement an ORAM protocol from a search-pattern-hiding SSE scheme), yet this would only offer us a lower bound in  $\log K$  on the update computational complexity, while we are looking for something in the order of  $n_w \log N$  (the performance of an SSE scheme naively implemented using ORAM).

There are still subtle differences with the ORAM lower bound, in particular in the way it will be proven. Namely, suppose that the first query matched  $n_1$  entries. If the second query matches  $n_2 \neq n_1$  entries, the adversary will immediately learn that the queries are different, and hence that the result entries are different. Yet, if  $n_2 = n_1$ , this is not true anymore, and we have to hide the possible repetition of the first query in this case. Similarly, the server already knows that he will have to access at least  $n_w$  different entries during a search query on  $w$ : we do not have to hide this information. Also, as we are only interested in the result set, not in the result order, we might be able to batch some accesses.

Hence, we will have to consider the number of entries that have been previously matching a search query, with a small caveat: for the  $i + 1$ -th search query, the query being done on keyword  $w$ , we are only interested in entries matching a keyword in the set  $\{w_j | 1 \leq j \leq i \text{ and } n_{w_j} \neq n_w\}$ , *i.e.* the set of previously searched keywords matching a number of documents different from the one of  $w$ . Hence, for a partial history  $H_i = (\text{DB}, w_1, \dots, w_i)$ , we denote by  $\overline{N}(H_i, w)$  this quantity, and

$$\overline{N}(H_i, w) = |\text{DB}| - \sum_{\substack{j=1 \\ |\text{DB}(w_j)| \neq |\text{DB}(w)|}}^i |\text{DB}(w_j)|.$$

**Theorem 3.2.** *Let  $\Sigma$  be a (static) SSE scheme that is  $\mathcal{L}$ -indistinguishably secure, with  $\mathcal{L} \preceq \mathcal{L}_{\overline{\text{sp}}}$ , with a client state  $\sigma$  of size  $|\sigma|$ . Then the overall computational complexity of the Search protocol (the sum of the computational complexity for the client and the server) for a keyword  $w$  matching  $n_w$  documents, after the execution of the history  $H$  is*

$$\Omega \left( \frac{\log \binom{\overline{N}(H, w)}{n_w}}{\log |\sigma| \cdot \log \log \binom{\overline{N}(H, w)}{n_w}} \right).$$

The proof will be based on similar ideas to the ORAM lower bound result of [GO96]. We will see that, in some sense, the  $\log \log$  factor is an artifact of the proof. The case of the  $\log \left( \frac{N(H,w)}{n_w} \right)$  expression instead of  $n_w \log N$  is a bit more subtle.

We start the proof with the following lemma, which is an adaptation of the ORAM lower bound to the searchable encryption setting.

**Lemma 3.3.** *Let  $\Sigma$  be a (static) SSE scheme that is  $\mathcal{L}$ -indistinguishably secure, with  $\mathcal{L} \preceq \mathcal{L}_{\overline{\text{sp}}}$ . Suppose that the Search algorithm uses at most  $b$  memory blocks to run the query on  $w$  ( $b$  can depend on the history). Then, the computational complexity of the search protocol is*

$$\Omega \left( \frac{\log \left( \frac{\overline{N}(H,w)}{n_w} \right)}{\log b(b+2)} \right).$$

*Proof.* Without loss of generality, we can suppose that the adversary only gets to see the read (and write) accesses to the encrypted database generated by the search protocol: this will only reduce his capabilities. We also only consider a non-adaptive attacker who chooses two fixed histories. In this setting, the SSE indistinguishability game can be modeled as follows:

- a player, who can hold at most  $b$  balls, makes (probabilistic) accesses to the encrypted database to answer the sequence of  $t$  search queries corresponding to the keywords  $(w_1, \dots, w_t)$  on the database DB;
- an observer, who gets to see the accesses.

The observer/adversary will win the game if he is able to distinguish the execution of two search sequences  $(w_1, \dots, w_t)$  and  $(w'_1, \dots, w'_t)$ , respectively on the databases DB and DB' such that  $|\text{DB}| = |\text{DB}'|$  and  $|\text{DB}(w_i)| = |\text{DB}'(w'_i)|$  for  $i = 1, \dots, t$ .

The encrypted database is modeled as  $N$  atomic entries, each encoding a document/keyword pair of the dataset, equivalent to the balls in the ORAM lower bound proof of Goldreich and Ostrovsky [GO96]. These entries are stored in non-transparent cells holding a single entry. At any time the player accesses a cell, he can either fetch the entry residing in this cell, place an entry in it, or do nothing. The observer will see that the player accessed this cell, but not what he just did with it. It is important to note that it is not because the scheme is static that the encrypted database cannot be modified by the Search algorithm: the fact that the scheme is static only implies that it does not support Update operations.

To answer these  $t$  search queries, the player will make a sequence of  $q$  *visible* accesses  $V = (v_1, \dots, v_q)$ , observable by the adversary. For each accesses  $v_i$ , the player will perform a *hidden action*  $h_i$ , which the observer cannot see. As mentioned before, the player can take an entry from the cell, place an entry in the cell or do nothing. In particular, there are  $b + 2$  possible actions ( $b$  ‘placing’ actions, the ‘taking’ action, and the ‘nothing’ action). This action sequence  $(v_1, h_1), \dots, (v_q, h_q)$  satisfies the search queries sequence  $(w_1, \dots, w_t)$  if and only if there exists a sequence  $1 = j_0 \leq j_1 \leq \dots \leq j_t \leq q$  such that for every  $i$ -th search request, the player has held every entry corresponding to the entries  $(w_i, \text{ind})$ , for  $\text{ind} \in \text{DB}(w_i)$ , after the actions  $(v_{j_{i-1}+1}, h_{j_{i-1}+1}), \dots, (v_{j_i}, h_{j_i})$ . We note  $\delta q_i = j_i - j_{i-1} + 1$ .

Let us focus on the  $i$ -th search request. As the player holds at most  $b$  entries, a fixed sequence  $(v_j, h_j), \dots, (v_{j+\delta q}, h_{j+\delta q})$  can satisfy at most  $b^{\delta q}$  different search queries (a very large upper bound on the number of entries that the player can fetch in  $\delta q$  actions). Also, each  $\delta q$ -long visible accesses



sub-sequence  $(v_j, \dots, v_{j+\delta q_i})$  may be coupled with  $(b+2)^{\delta q_i}$  hidden action sequences. In the end, we can say that each sub-sequence  $\delta V_i$  of length  $\delta q_i$  can satisfy at most  $b^{\delta q_i} (b+2)^{\delta q_i}$  search queries.

Finally, as the search pattern is hidden, but the size pattern is revealed, the number of search queries matching  $n$  documents in the database DB after the search queries  $w_1, \dots, w_i$  is

$$\binom{\overline{N}(H, w)}{n},$$

where  $H = (\text{DB}, w_1, \dots, w_i)$  is the history of previous executions. The reason for this non-trivial expression is first that the order in which the entries are fetched does not matter (we are only interested in the result set), then that the player does not have to hide that he does not access the entries that were previously fetched for searches matching a different number of results. Namely, after the execution of the history  $H$ , there are exactly  $\overline{N}(H, w)$  non-touched entries matching  $n \neq n_w$  entries. As the  $i$ -th sub-sequence  $V_i$ , of length  $\delta q_i$  must satisfy  $\overline{N}(H_{i-1}, w_i)$  possible search queries, we have

$$b^{\delta q_i} (b+2)^{\delta q_i} > \binom{\overline{N}(H_{i-1}, w_i)}{n_{w_i}},$$

and, as a consequence,

$$\delta q_i = \Omega \left( \frac{\log \binom{\overline{N}(H_{i-1}, w_i)}{n_{w_i}}}{\log b(b+2)} \right).$$

□

Note that the expression  $\binom{\overline{N}(H_{i-1}, w_i)}{n_{w_i}}$  might look a little artificial: we suppose that the player will access the matching entry in any order, while, in practice, he will chose one and never deviate from it, pledging for a term  $\frac{\overline{N}(H_{i-1}, w_i)!}{n_{w_i}!}$  instead of the binomial expression. There is a little caveat to this remark, though: the entries corresponding to the documents matching the search query will have to be (randomly) relocated once they have been accessed (leaking the search pattern otherwise), and the player can optimize the number of visible accesses he makes, which he could not do with ORAM – remember that it is not a problem to leak that different entries/balls were accessed during a search query as this information is obvious.

In the main proof of the theorem, we will also use the following technical lemma.

**Lemma 3.4.** *Let  $C > 8$ ,  $D \geq 0$  and  $f : [D+1, +\infty) \rightarrow [1, +\infty)$  defined as*

$$f(x) = \max \left\{ x - D, \frac{C}{\log x(x+2)} \right\}.$$

*Then for all  $x \in [D+1, +\infty)$ ,*

$$\frac{C}{4 \log(D+2) \cdot \log C} - 1 < f(x).$$

*Proof.* This lemma can be shown using simple analysis. We start by a simple variable change:

$$\max_{x \in [D+1, +\infty)} \left\{ x - D, \frac{C}{\log x(x+2)} \right\} = \max_{x \in [1, +\infty)} \left\{ x, \frac{C}{\log(x+D)(x+D+2)} \right\}$$

Then, notice that, that, as  $\log$  is an increasing function,

$$\frac{C}{\log(x+D)(x+D+2)} \geq \frac{C}{2\log(x+D+2)} = \frac{C}{2(\log(x/D'+1) + \log(D'))}$$

where  $D' = D + 2$ . Also, by denoting  $C' = \frac{C}{2\log D'}$ , we have that

$$\frac{C}{\log(x+D)(x+D+2)} \geq \frac{C'}{\log_{D'}(x/D'+1) + 1} \geq \frac{C'}{\log_{D'}(x+1) + 1}$$

again, because  $\log$  is an increasing function. As a consequence

$$\max_{x \in [D+1, +\infty)} \left\{ x - D, \frac{C}{\log x(x+2)} \right\} \geq \max_{x \in [1, +\infty)} \left\{ x, \frac{C'}{\log_{D'}(x+1) + 1} \right\}$$

Let  $g(x) = \max \left\{ x, \frac{C'}{\log_{D'}(x+1) + 1} \right\}$ . Lower bounding  $f$  on  $[D+1, +\infty)$  is equivalent to lower bounding  $g$  on  $[1, +\infty)$ . Let us separate two cases, whether  $x \geq D' - 1$  or  $x < D' - 1$ .

If  $x < D' - 1$ , then  $\log_{D'}(x+1) + 1 \leq 2$  and

$$g(x) \geq \max \{x, C'/2\} \geq \frac{C}{4\log D'}.$$

If  $x \geq D' - 1$ , then  $\log_{D'}(x+1) + 1 \leq 2\log_{D'}(x+1)$  and

$$g(x) \geq \max \left\{ x, \frac{C'/2}{\log_{D'}(x+1)} \right\}.$$

Also, as  $x \mapsto x$  is an increasing function and  $x \mapsto \frac{C'}{2\log_{D'}(x+1)}$  is decreasing, the minimum is reached when  $x = \frac{C'}{2\log_{D'}(x+1)}$ . Finding this minimum is equivalent to finding  $x^*$  such that  $h(x^*) = 0$  where

$$h(x) = x - \frac{C'}{2\log_{D'}(x+1)} = x - \frac{C}{4\log(x+1)}$$

as  $C' = \frac{C}{2\log D'}$ . In particular,  $f(x) \geq x^*$ . Note that we are actually only looking for a lower bound of  $x^*$ , not its exact value. Hence, as  $h$  is a continuous and strictly increasing function on  $[1, +\infty)$ ,  $x^*$  is the only value on which  $h$  annihilates and  $h(x) \leq 0 \Leftrightarrow x \leq x^*$ . Also, we have that

$$(x+1) - \frac{C}{4\log(x+1)} \leq 0 \Rightarrow x \leq x^*,$$

Let  $C'' = C/4$  and suppose that  $\log C'' > \log \log C'' \geq 0$ . Then, we directly have that

$$\frac{1}{\log C''} - \frac{1}{\log \frac{C''}{\log C''}} = \frac{1}{\log C''} - \frac{1}{\log C'' - \log \log C''} \leq 0,$$

and for  $x_0 = \frac{C''}{\log C''} - 1$ , we have that  $(x_0 + 1) - \frac{C}{4\log(x_0+1)} \leq 0$ , and then that  $x_0 \leq x^*$ .

To conclude, we just have to check that  $\log C'' > \log \log C'' \geq 0$ . The last inequality is verified for  $C \geq 8$ . For the first inequality, we have to study  $t : x \mapsto x - \log x$  on the interval  $[1, +\infty)$ . Its

derivative  $x \mapsto 1 - \frac{1}{x \cdot \ln 2}$  (where  $\ln$  is the natural logarithm) is negative on  $[1, \frac{1}{\ln 2}]$  and positive on  $[\frac{1}{\ln 2}, +\infty[$ . Thus,  $t$  reaches its minimum for  $x = \frac{1}{\ln 2} \approx 1.443$ , and

$$t(x) \geq t\left(\frac{1}{\ln 2}\right) \approx 0.914 > 0.$$

As a consequence, and because we supposed that  $C > 8$ ,  $\log C'' > 0$ , and  $\log C'' > \log \log C''$ , we have that  $x_0 \geq \frac{C}{4(\log C) - 2} - 1 \geq \frac{C}{4 \log C} - 1$ .

We can conclude the proof of this lemma by stating that

$$\max_{x \in [D+1, +\infty)} \left\{ x - D, \frac{C}{\log x(x+2)} \right\} \geq \min \left\{ \frac{C}{4 \log D}, \frac{C}{4 \log C} - 1 \right\} \geq \frac{C}{4 \log(D+2) \cdot \log C} - 1$$

□

The proof of the theorem is now very easy, using the previous lemmata.

*Proof of Theorem 3.2.* The memory  $b$  used during the execution of the Search protocol can be separated into two parts. First, there is the client's state, and then the additional memory  $b'$  used by either the client or the server to execute the protocol itself:  $b = |\sigma| + b'$ . Without loss of generality, we can suppose that the Search protocol touches each of the  $b'$  memory blocks at least once: this is not true for a block, then the protocol does not need it. Hence, the complexity of the protocol is at least  $\Omega(b')$ .

By combining this first lower bound with Lemma 3.3, we have that the computational complexity of a search query is

$$\Omega \left( \max \left\{ b - |\sigma|, \frac{\log \left( \frac{\overline{N}(H,w)}{n_w} \right)}{\log b(b+2)} \right\} \right).$$

We can immediately conclude using Lemma 3.4. □

We can improve this bound if we suppose that  $\left( \frac{\overline{N}(H,w)}{n_w} \right)$  is large enough. Indeed, we can show, in a similar way to Lemma 3.4, that

$$\max \left\{ x - D, \frac{C}{\log x(x+2)} \right\} > \frac{C}{4 \log(D+2) \cdot \log \log C}$$

if  $\log C/4 > \log \log \log C/4 > 0 \Leftrightarrow C/4 > 2^{2^0} \Leftrightarrow C > 16$ . Indeed, more generally, for all  $a \in \mathbb{N}^*$ ,

$$\max \left\{ x - D, \frac{C}{\log x(x+2)} \right\} > \frac{C}{4 \log(D+2) \cdot \overbrace{\log^{(a)} C}^{a \text{ times } 2^0}} \text{ if } C > 4 \cdot \overbrace{2^{2^{\dots^{2^0}}}}^{a \text{ times}}.$$

where  $\log^{(1)} x = \log x$  and  $\log^{(a)} x = \log \log^{(a-1)} x$  for  $a > 1$ . Hence, for any for all  $a \in \mathbb{N}^*$ , and large enough  $\left( \frac{\overline{N}(H,w)}{n_w} \right)$ , the complexity of a search query is

$$\Omega \left( \frac{\log \left( \frac{\overline{N}(H,w)}{n_w} \right)}{\log |\sigma| \cdot \log^{(a)} \left( \frac{\overline{N}(H,w)}{n_w} \right)} \right).$$

This leads us to believe that the  $\log \log$  factor in denominator of the lower bound in Theorem 3.2 is an artifact of the proof, and that the actual lower bound is

$$\Omega \left( \frac{\log \binom{\bar{N}(H,w)}{n_w}}{\log |\sigma|} \right).$$

We can also derive an immediate corollary from Theorem 3.2.

**Corollary 3.5.** *The complexity of the first query of an SSE scheme,  $\text{DB}$  being the database,  $\sigma$  the client's state, and  $w$  the searched keyword is*

$$\Omega \left( \frac{\log \binom{N}{n_w}}{\log |\sigma| \cdot \log \log \binom{N}{n_w}} \right).$$

*Proof.* For the first search query, there are no previous request and  $\bar{N}(H_0, w) = N$  □

It is interesting to see that this lower bound does not hold anymore when the leakage is even slightly increased. Namely, suppose that besides  $N$ , the setup also leaks, for each  $n \in \mathbb{N}$ , the number  $K_n$  of keywords  $w$  such that  $|\text{DB}(w)| = n$ . We can then design a scheme  $\Sigma$ , optimal in terms of server storage, and whose search complexity for a keyword  $w$  is  $\log K_{n_w} + n_w$ .

For each  $n$  such that  $K_n \neq 0$ ,  $\Sigma$  initializes an ORAM of size  $K_n$  where each block corresponds to a keyword  $w$  such that  $|\text{DB}(w)| = n$  and stores  $\text{DB}(w)$ . Also, for each keyword  $w \in K$ , the client stores  $n_w$ . Then a search query only consists in reading the block corresponding to the search keyword  $w$  in the ORAM for keywords with  $n_w$  results. With state of the art ORAM (e.g. [SDS+13a]), this only requires  $\log K_{n_w} + n_w$  operations.

**Communication vs computation complexity.** Theorem 3.2 states a lower bound on the total computational complexity of the client and the server during a search query, but does not state how the work is shared between the two. This is similar to the ORAM lower bound of [GO96], as explained in [DDF+16]. Indeed, in the setting of [GO96], the server is not allowed to do computations, so we end up with a lower bound on the bandwidth overhead. But once, the server is allowed to 'help' the client, this bandwidth overhead does not hold anymore, but the total number of atomic operations still has to satisfy the lower bound.

We are exactly in the same case with our SSE lower bound: if the server is supposed to be completely passive (i.e. act like some basic storage device), our computational lower bound will also be a communication lower bound. On the other end, the searchable encryption scheme could be implemented as a garbled RAM program (see [LO13b]) executed by the server on its own: the size of the search token (i.e. the garbled program) would only depend on the security parameter, and there would not be any bandwidth overhead, whereas the computational overhead is very high.

### 3.3 The Locality of Searchable Encryption

Although some searchable encryption schemes are asymptotically optimal in terms of efficiency, (e.g. [CGKO06; CJJ+14; Bos16]) in practice, their performance is worse than unencrypted systems by an order of magnitude, an often poorly scale to very large databases. Indeed, the bottleneck of these schemes is not the cryptography but the throughput of the external storage where the encrypted database is stored. Namely, for the search protocol, these schemes make one random access per

match of the query, while unencrypted databases organize the data so that all the results can be fetched in a constant number of accesses.

Designing searchable encryption schemes with high storage locality — *i.e.* a scheme making a small number of memory accesses during the search protocol — is a major challenge, and is necessary to make encrypted database scalable. One could argue that using legacy-compatible construction would solve this problem, but such systems barely provide any security. Also, a scheme that returns the entire encrypted database upon a search query would have perfect locality but terrible performance. Finally, Chase and Kamara [CK10] proposed a scheme that has perfect locality, reads no more information than needed during a search query, but has very large encrypted databases: the server needs to store  $K \cdot \max_{w \in W} n_w$ , which can be much larger than  $N$  (up to  $K \cdot N$ ).

Cash and Tessaro actually presented a lower bound on the locality of searchable encryption schemes in [CT14]: a scheme leaking at most the size of the database, the number of distinct keywords, the number of documents, the search pattern and the number of results of a search query cannot be both storage optimal ( $|\text{EDB}| = \mathcal{O}(N)$ ), have constant locality (the number of sequential accesses to the encrypted database) and achieve constant read efficiency (the amount of data read during a search query is linear in the number of results).

### 3.3.1 Locality, Read Efficiency and Overlapping Reads

Here, we quickly recall the definitions introduced in [CT14] so that we can formally state the lower bound result.

**Definition 3.8** (Read Pattern). *We interpret EDB as a bit string of size  $M$ , and  $\text{EDB}[a, b]$  is the substring starting by the  $a$ -th and ending with the  $b$ -th bit. On the server side, the  $\text{Search}(K_\Sigma, w; \text{EDB})$  protocol adaptively computes a sequence of intervals  $([a_1, b_1], \dots, [a_n, b_n])$  where  $[a_1, b_1]$  depends on the first message of the client,  $[a_2, b_2]$  on previously received messages and on  $\text{EDB}[a_1, b_1]$ , etc.*

*The read pattern  $\text{RdPat}(w, \text{EDB})$  is this sequence  $([a_1, b_1], \dots, [a_n, b_n])$  of intervals.*

From the read pattern, it is now easy to define the locality.

**Definition 3.9** (Locality). *A searchable encryption scheme  $\Sigma$  is  $r$ -local (has locality  $r$ ) if for any  $\lambda$ ,  $\text{DB}$  and  $w \in W$ ,  $\text{RdPat}(w, \text{EDB})$  consists of at most  $r$  intervals when  $(K_\Sigma, \text{EDB}) \stackrel{\$}{\leftarrow} \text{Setup}(1^\lambda, \text{DB})$ . If  $r = 1$ , we say that  $\Sigma$  has perfect locality.*

Note that  $r$  can depend on  $\lambda$  or on  $|\text{DB}|$ .

**Definition 3.10** (Read Efficiency). *A searchable encryption scheme  $\Sigma$  is  $c$ -read efficient (has read efficiency  $c$ ) if for any  $\lambda$ ,  $\text{DB}$  and  $w \in W$ ,  $\text{RdPat}(w, \text{EDB})$  consists of intervals of total length at most  $c \cdot |\text{BinEnc}(\text{DB}(w))|$  bits, where  $\text{BinEnc}(\text{DB}(w))$  is the binary representation of  $\text{DB}(w)$ , *i.e.* the concatenation of all indices represented as bit strings. Without loss of generality, we can suppose that  $|\text{BinEnc}(\text{DB}(w))| = \ell |\text{DB}(w)|$ , as we encode indices over  $\ell$  bits (cf. Section 3.1.1).*

The lower bound of Cash and Tessaro uses a restricted version of the read efficiency: they look at the number of new bits read during the search query. Hence, they define the notion of overlapping reads, formalized as follows.

**Definition 3.11** (Overlapping Reads). *A searchable encryption scheme  $\Sigma$  has  $\alpha$ -overlapping reads if for all  $\lambda$ , and all  $\text{DB}$ , the read pattern induced by the search of each keyword in  $\text{DB}$  has an overlap of at most  $\alpha$  bits with the read patterns of all other (previous or future) keywords (with probability 1 over*

the computation of  $(K_\Sigma, \text{EDB}) \xleftarrow{\$} \text{Setup}(1^\lambda, \text{DB})$  and the executions of the Search protocol.) When  $\alpha = 0$ , we say that  $\Sigma$  has disjoint reads.

The value  $\alpha$  can be independent of  $N$ , but may depend on  $\lambda$  or on  $K$ , e.g. when  $\Sigma$  has to read a keyword-indexed table storing meta-data.

### 3.3.2 A Lower Bound on the Locality of Searchable Encryption

Now that all these notions are defined, we can formally state the negative result of Cash and Tessaro. However, their result stands only for a minimal level of leakage: legacy-compatible constructions using deterministic encryption can regroup entries and achieve perfect locality, read efficiency and server storage. Hence, we define  $\mathcal{L}_{loc}$  as

$$\begin{aligned} \mathcal{L}_{loc}^{\text{Stp}}(\text{DB}) &= \left( N, K, D, \max_{w \in \mathcal{W}} |\text{DB}(w)| \right), \\ \mathcal{L}_{loc}^{\text{Srch}}(w) &= \text{DB}(w). \end{aligned}$$

**Theorem 3.6** (Theorem 4.1 of [CT14]). *Let  $\Sigma$  be a  $\mathcal{L}$ -indistinguishably secure searchable encryption scheme, with  $\mathcal{L} \preceq \mathcal{L}_{loc}$ . If  $\Sigma$  has locality  $r$  and  $\alpha$ -overlapping reads, then it has  $\omega\left(\frac{|\text{BinEnc}(\text{DB})|}{r(\alpha+1)}\right)$  server storage. In particular, if  $\Sigma$  has perfect locality, perfect read efficiency, and disjoint reads, then  $\Sigma$  has  $\omega(|\text{BinEnc}(\text{DB})|)$  server storage.*

The proof of this theorem in [CT14] only requires the non-adaptive security of  $\Sigma$ . Also note that the last case – perfect read efficiency and disjoint reads – implies that  $\Sigma$  leaks the search pattern  $\text{sp}$ .

The intuition behind this result is that, with a perfectly local and read efficient scheme, after a sequence of searches, the server can look which parts of the encrypted database were accessed and not accessed. Hence, if there is a keyword, not previously queried, and matching many documents, the server will see a large interval of untouched bits in EDB, which is not the case (in average) if there only are keywords with a small number of matches. In particular, the server would be able to distinguish these two kinds of histories, despite them having the same  $\mathcal{L}_{loc}$  leakage. We refer to [CT14] for the complete proof and the extension to the more general case of non-perfect locality and read efficiency.

### 3.3.3 Constructions with Improved Locality

Besides their lower bound result, Cash and Tessaro present a construction of a static SSE scheme with improved locality, namely  $\mathcal{O}(\log N)$  locality, perfect read efficiency and  $\mathcal{O}(N \log N)$  server storage. The lower bound of Theorem 3.6 was later shown (essentially) tight by Asharov *et al.* in [ANSS16]. In this work, the authors present three constructions with perfect read efficiency, based on allocation algorithms. The first two schemes have optimal storage, and have read efficiency of, respectively,  $\mathcal{O}(\log N \cdot \log \log N)$  and  $\mathcal{O}(\log \log N \cdot \log \log \log N)$ , the second one assuming that keywords do not match more than  $N^{1-1/\log \log N}$  documents. The last scheme is an improvement over the Cash and Tessaro construction which achieves perfect locality and read efficiency at the cost of  $\mathcal{O}(N \log N)$  server storage.

The practicality of these constructions was studied by Demertzis and Papamanthou in [DP17]. Despite being asymptotically efficient, the perfectly local constructions of Asharov *et al.* are not better in practice than the non-local  $\Pi_{\text{bas}}$  scheme of [CJJ+13]. The authors hence present new constructions,

with  $\mathcal{O}(L)$  locality,  $\mathcal{O}(N^{1/s}/L)$  read efficiency and  $\mathcal{O}(N \cdot s)$  storage, hence asymptotically worse than the first two schemes of [ANSS16], but that perform practically better by an order of magnitude.

The first dynamic searchable encryption schemes with improved locality have also been presented in [DP17]. Their idea is to build dynamic SE from static schemes by perpetually rebuilding the encrypted database, and including the newly inserted records. More precisely, the encrypted database consists of levels of exponential size — level  $i$  contains  $2^i$  records — each level being an encrypted database of a static scheme. Initially, all the levels are empty. When a new item is inserted in the database, either level 0 is empty, and the entry is encrypted and placed there, or it is already full, and then the client downloads and empties the first  $\ell$  levels,  $\ell$  being the first empty level, and encrypts these  $2^\ell - 1$  entries plus the newly inserted entry into the  $\ell$ -th level. This idea is borrowed from the hierarchical-ORAM literature (e.g. [GM11; GMOT12]) and was already applied to searchable encryption in [SPS14]. It is fully described in Section 4.4.

### 3.4 Leakage Abuse Attacks

In Section 3.1.3, we saw security definitions for the confidentiality of searchable encryption based on leakage functions. The semantic of these definitions is that nothing beyond the leakage is learned by the server. Yet, this does not prevent the leakage function itself to leak a lot of information.

For example, we could define the leakage function  $\mathcal{L}_{all}$  as follows:

$$\begin{aligned}\mathcal{L}_{all}^{\text{Stp}}(\text{DB}) &= \text{DB}, \\ \mathcal{L}_{all}^{\text{Srch}}(q) &= q, \\ \mathcal{L}_{all}^{\text{Updt}}(\text{op}, \text{in}) &= (\text{op}, \text{in}).\end{aligned}$$

This leakage function leaks *everything*, and any searchable encryption scheme is  $\mathcal{L}_{all}$ -adaptively-semanticly secure.

Understanding the impact of the leakage of a scheme is hence very important, and building a hierarchy of strictly increasing leakage functions — in the sense of Definition 3.6 — is crucial.

At the same time, we must be able to assess the practical security of the leakage functions, which is slightly different. Namely the leakage class will tell us which leakage functions are equivalent in general, while devising attacks breaking the confidentiality of schemes only from its leakage will show that this particular leakage function might leak more than expected in some cases. We will see that this is particularly the case for active attacks, where the adversary can insert some known or chosen documents in the database. Also, in many cases, additional information about the database (e.g. the distribution of keywords) can help the server to recover some information about the queried keywords. The next two sections present some results relative to these *leakage-abuse attacks* against single-keyword searchable encryption. We do not focus here on legacy-compatible constructions, as this study mainly focuses on *ad hoc* schemes. We just underline that many of those constructions are prone to frequency analysis, as described by Naveed, Kamara and Wright [NKW15].

Note that in Chapter 7, we will analyze these leakage abuse attacks in more details.

#### 3.4.1 Attacks Based on Keyword Frequency

The first kind of attacks that have been developed by the community is the set of attacks based on keyword frequency associated with prior knowledge about the database. This line of work started with a paper by Islam, Kuzu, and Kantarcioglu [IKK12]. The adversary uses the known distribution  $\mathcal{D}$  of the co-occurrence matrix of the targeted set of keywords. This matrix maps to each pair of

keywords the number of documents in which both keywords appear. From the observation of the documents access pattern (the list of result indices), the adversary builds a co-occurrence matrix that follows the distribution  $\mathcal{D}$ , up to rows and columns permutation. Using simulated annealing, he will find this permutation, which will be the match between queries and keywords. The issue with this attack is that it does not scale well with the number of keywords, as shown by Cash *et al.* in [CGPR15].

The count attack of [CGPR15] also uses a co-occurrence matrix, but uses it only to leverage some prior knowledge issued from the uniqueness of the number of results for some keywords: there are some keywords  $w$  such that no other keyword  $w'$  match the exact same number of documents. Hence, for the full or partial knowledge of the database, the adversary is able to decrypt the queries keyword. For this initial knowledge, using co-occurrence information, as in [IKK12], the adversary will be able to retrieve the queries whose result count is not unique.

### 3.4.2 File Injection Attacks

All the previously described attacks are both passive and non-adaptive: they do not require the attacker to interact with the encrypted database, or with the client in general. In particular, they do not use the new attack means introduced by dynamic schemes, such as inserting specifically crafted documents in the database.

*File injections attacks* do exactly that. Formally speaking they were introduced by Cash *et al.* in [CGPR15] to attack legacy-compatible schemes. They were used against *ad hoc* schemes by Zhang, Katz, and Papamanthou [ZKP16]. The basic idea of their attack is that the server can insert documents with half of the keywords in  $W$ . Then, the attacker will observe which of these inserted documents match the search query issued by the client (*e.g.* using the access pattern of the search algorithm). Each of the crafted documents brings one bit of information (whether the query matches the document or not), and using a standard binary search technique, the server can easily find which keyword was found. Although, some countermeasures exist (such as limiting the number of keywords by documents), some adaptive variants are extremely powerful — we will present these variants and countermeasures in Chapter 4.

## References

- [ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. *Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations*. In: *48th ACM STOC*. Ed. by Daniel Wichs and Yishay Mansour. ACM Press, June 2016, pp. 1101–1114 (cit. on pp. 58, 59, 69, 71).
- [BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O’Neill. *Deterministic and Efficiently Searchable Encryption*. In: *CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. LNCS. Springer, Heidelberg, Aug. 2007, pp. 535–552 (cit. on p. 41).
- [BDOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. *Public Key Encryption with Keyword Search*. In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 506–522 (cit. on p. 41).
- [BKOS07] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. *Public Key Encryption That Allows PIR Queries*. In: *CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. LNCS. Springer, Heidelberg, Aug. 2007, pp. 50–67 (cit. on p. 41).



- [Bos16] Raphael Bost. *Σοφός: Forward Secure Searchable Encryption*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1143–1154 (cit. on pp. [viii](#), [9](#), [10](#), [12](#), [56](#), [65](#)).
- [CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions*. In: *ACM CCS 06*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, Oct. 2006, pp. 79–88 (cit. on pp. [9](#), [42–45](#), [56](#), [72](#), [152](#)).
- [CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. *Leakage-Abuse Attacks Against Searchable Encryption*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 668–679 (cit. on pp. [60](#), [66](#), [151–153](#), [156](#), [161](#), [162](#), [170–172](#)).
- [CJJ+13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 353–373 (cit. on pp. [9](#), [11](#), [43](#), [45](#), [58](#)).
- [CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. [10](#), [49](#), [56](#), [73](#), [83](#), [99](#), [129](#), [163](#), [170](#), [171](#)).
- [CK10] Melissa Chase and Seny Kamara. *Structured Encryption and Controlled Disclosure*. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 577–594 (cit. on pp. [9](#), [12](#), [43](#), [45](#), [57](#)).
- [CM05] Yan-Cheng Chang and Michael Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*. In: *ACNS 05*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. LNCS. Springer, Heidelberg, June 2005, pp. 442–455 (cit. on pp. [9](#), [42](#)).
- [CT14] David Cash and Stefano Tessaro. *The Locality of Searchable Symmetric Encryption*. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 351–368 (cit. on pp. [57](#), [58](#), [68](#)).
- [DDF+16] Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. *Onion ORAM: A Constant Bandwidth Blowup Oblivious RAM*. In: *TCC 2016-A, Part II*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9563. LNCS. Springer, Heidelberg, Jan. 2016, pp. 145–174 (cit. on pp. [7](#), [56](#)).
- [DP17] Ioannis Demertzis and Charalampos Papamanthou. *Fast Searchable Encryption With Tunable Locality*. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM. 2017, pp. 1053–1067 (cit. on pp. [58](#), [59](#), [68](#), [69](#)).
- [GM11] Michael T. Goodrich and Michael Mitzenmacher. *Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation*. In: *ICALP 2011, Part II*. Ed. by Luca Aceto, Monika Henzinger, and Jiri Sgall. Vol. 6756. LNCS. Springer, Heidelberg, July 2011, pp. 576–587 (cit. on pp. [59](#), [71](#), [130](#), [133](#), [147](#)).
- [GMOT12] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. *Privacy-preserving group data access via stateless oblivious RAM simulation*. In: *23rd SODA*. Ed. by Yuval Rabani. ACM-SIAM, Jan. 2012, pp. 157–167 (cit. on pp. [59](#), [71](#)).

- [GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. *TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption*. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563–592 (cit. on pp. 7, 8, 45, 69, 73, 99, 101, 161, 163, 170).
- [GO96] Oded Goldreich and Rafail Ostrovsky. “Software protection and simulation on oblivious RAMs”. In: *Journal of the ACM* 43.3 (1996), pp. 431–473 (cit. on pp. 7, 8, 51, 52, 56, 82).
- [Goh03] Eu-Jin Goh. *Secure Indexes*. Cryptology ePrint Archive, Report 2003/216. <http://eprint.iacr.org/2003/216>. 2003 (cit. on pp. 9, 11, 42).
- [Gol04] Oded Goldreich. *Foundations of cryptography*. Cambridge University Press, 2004 (cit. on pp. 29, 44, 47).
- [IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. *Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation*. In: *NDSS 2012*. The Internet Society, Feb. 2012 (cit. on pp. 59, 60, 151, 152, 173).
- [KO12] Kaoru Kurosawa and Yasuhiro Ohtaki. *UC-Secure Searchable Symmetric Encryption*. In: *FC 2012*. Ed. by Angelos D. Keromytis. Vol. 7397. LNCS. Springer, Heidelberg, Feb. 2012, pp. 285–298 (cit. on pp. 9, 10, 47).
- [KO13] Kaoru Kurosawa and Yasuhiro Ohtaki. *How to Update Documents Verifiably in Searchable Symmetric Encryption*. In: *CANS 13*. Ed. by Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab. Vol. 8257. LNCS. Springer, Heidelberg, Nov. 2013, pp. 309–328 (cit. on pp. 10, 47).
- [LO13b] Steve Lu and Rafail Ostrovsky. *How to Garble RAM Programs*. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 719–734 (cit. on pp. 8, 56).
- [NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. *Inference Attacks on Property-Preserving Encrypted Databases*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 644–655 (cit. on pp. 10, 12, 59).
- [SDS+13a] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. *Path ORAM: an extremely simple oblivious RAM protocol*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 299–310 (cit. on pp. 7, 56, 82).
- [SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. viii–x, 10, 13, 45, 59, 67, 71, 73, 81, 97–99, 115, 119, 122, 130–133, 141, 147, 163, 170).
- [SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. *Practical Techniques for Searches on Encrypted Data*. In: *2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2000, pp. 44–55 (cit. on pp. 8, 9, 42).
- [ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. *All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption*. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 2016, pp. 707–720 (cit. on pp. 10, 12, 60, 66, 76, 114, 152, 153, 156, 157).



You've got to always go back in time if you  
want to move forward.

SNOOP DOGG

# Forward Privacy 4

**D**YNAMISM IS A MUST-HAVE FEATURE for practical searchable encryption constructions: being able to add new documents to a dataset once it already has been encrypted and outsourced is an essential functionality that must be implemented by state-of-the-art schemes. Unfortunately, as it is often the case in cryptography, new features imply new attack means for the adversary. Namely, in a static database setting, it is hard for an adversary to influence the dataset *a priori*, before he gets to see its encryption or any search query. In the contrary, in the dynamic setting, he might make the client insert in the database documents that would help him to get information about future or even past search queries.

In this chapter, we will study these *file injection attacks*, their consequences on the security of dynamic searchable encryption schemes, and ways to mitigate them. In particular, we will define the notion of *forward privacy*, which ensures that updates to the database are oblivious, and leak no information about the keywords matching the modified documents, thwarting the most devastating versions of the file injection attacks. We will see that forward privacy comes at an unavoidable cost by presenting heuristics and a lower bound on the efficiency of forward private schemes. Then, we describe the ideas behind the first scheme that was designed to achieve this notion of forward privacy, SPS, which uses techniques closely related to Oblivious RAM, and hence inherits their inefficiencies. Finally, we will construct two forward private schemes, which overcome these issues,  $\Sigma\phi\phi\phi$  and Diana, respectively from trapdoor permutations and range-constrained pseudorandom functions, the latter being constructed from regular pseudo-random functions.

Most of the results of this chapter have been published in [Bos16] and [BMO17].

## Contents

---

<b>4.1</b>	<b>File Injection Attacks</b>	<b>66</b>
<b>4.2</b>	<b>Definition of Forward Privacy</b>	<b>67</b>
<b>4.3</b>	<b>Constraints Induced by the Forward Privacy</b>	<b>68</b>
4.3.1	Constraints on Storage	68
4.3.2	Constraints on Locality	68
4.3.3	Constraints on Efficiency	69
<b>4.4</b>	<b>Building a Forward Private SSE Scheme from Static Schemes</b>	<b>71</b>
<b>4.5</b>	<b><math>\Sigma\phi\phi\phi</math>: Simple Optimal Forward Secure Searchable Encryption</b>	<b>72</b>
4.5.1	General Ideas	72
4.5.2	Basic Construction	74
4.5.3	Security	75
4.5.4	Derived Constructions	79
4.5.5	Reducing Client-side Storage	80
4.5.6	Comparison with Other Constructions	81

4.5.7	Outsourcing the Client's State . . . . .	81
4.6	<b>Diana: Forward-Secure SSE with Very Low Overhead</b> . . . . .	<b>82</b>
4.6.1	FS-RCPRF: Forward-Secure SSE from Range Constrained PRFs . . . . .	82
4.6.2	Diana, a GGM Instantiation of FS-RCPRF . . . . .	87
4.7	<b>Implementation and Evaluation of <math>\Sigma\phi\phi\varsigma</math> and Diana</b> . . . . .	<b>88</b>
4.7.1	Implementation Details . . . . .	88
4.7.2	Experimental Setting . . . . .	89
4.7.3	Results and Interpretation . . . . .	90

---

## 4.1 File Injection Attacks

As explained in Section 3.4.2, file injection attacks aim at breaking the confidentiality of the user's queries by injecting adversarially-controlled documents in the database. The first of these attacks was presented by Cash *et al.* in [CGPR15], but only targeted legacy-compatible encrypted databases, with much more leakage than what we usually consider in this thesis.

Zhang *et al.* [ZKP16] improved this attack against any dynamic scheme leaking, during a search query, the results of the search or when the matching documents have been inserted in the database. Their idea is that, if the adversary inserts a file containing half of the keyword set in the database, by checking if a search query matches this document or not, he will learn one bit of information about this keyword (which half of the keyword set the queried keyword belongs to). Note that, it suffices that the scheme leaks when the documents matching a query have been inserted (*e.g.* TimeDB( $w$ ) – see Section 3.2.2) for the server to get this bit of information.

By iteratively applying this technique, using a binary search algorithm, the adversary will be able to fully determine the queried keyword. More precisely, as described in [ZKP16], the attacker will generate  $d = \lceil \log K \rceil$  documents  $F_1, \dots, F_d$  such that  $F_i$  contains all the keywords whose  $i$ -th bit is set (we suppose that there is a mapping between  $W$  and  $\{1, \dots, K\}$ ).

This leads to a very efficient attack against a huge class of schemes (namely every scheme not based on ORAM), able to recover any query just by inserting  $\log K$  documents. This attack has a drawback though: it requires inserting documents with  $K/2$  keywords, which can be very large. The client might upper bound the number of keyword in a document by a parameter  $T$ , called the *threshold parameter* in [ZKP16]. This forces the attacker to increase a lot the number of documents necessary to recover any query. Indeed it grows from  $\log K$  to  $\frac{K}{2T} \log 2T$ . It is also quite easy to see that this is essentially optimal without any additional knowledge on the keyword distribution: inserting less documents would result in information incompleteness for some possible queries.

Now suppose that the scheme, when a new document is inserted, also leaks whether this new document matches a previous search query or not. The attacker can now mount his attack adaptively, in order to recover a previous query (the previous attack worked non-adaptively on queries issued *after* the file injections). In particular, it is only necessary to inject  $\frac{K}{T} + \log T$  documents for the attack to succeed. Again, it is easy to see that this is information-theoretically optimal when the attacker has no prior knowledge on the keywords and their distribution.

However, when he does and in particular when he knows the keyword distribution, the attacker can improve a lot the efficiency of the attack. Namely, as shown in [ZKP16], he only needs to insert  $\log 2T$  documents to adaptively recover a previous query, by reducing the number of candidates to  $2T$  using the number of matches of the query and the keyword frequency in the database.

These adaptive versions of the file injection attacks are efficient exactly because the insertions leak some information about the keywords contained in the new documents. To avoid such problem, we absolutely need the updates to be oblivious to the content of the documents, *i.e.* to the keywords. We call this property *forward privacy*.

Such behavior is also interesting in terms of functionality because it allows for the secure online build of the encrypted database. Indeed, the generic definition of SSE suppose that the encryption of the dataset, during the Setup algorithm, happens on the client side, and hence that, at some point, the client had enough local storage to hold the entire encrypted database (before sending it to the server). With forward privacy, this will no longer be necessary, as the encryption of the entire database can be done using multiple calls to the Update protocol, which will leak no information about the encrypted documents.

## 4.2 Definition of Forward Privacy

The notion of forward privacy for searchable encryption was introduced by Stefanov *et al.* in [SPS14]. The scheme the authors propose in that paper has the leakage  $\mathcal{L}_{\text{SPS}}$ , defined as follows:

$$\mathcal{L}_{\text{SPS}}^{\text{Srch}}(w) = (\text{sp}(w), \text{DB}(w), \text{AddDB}(w))$$

$$\text{and } \mathcal{L}_{\text{SPS}}^{\text{Updt}}(\text{op}, \text{ind}, \mathbf{w}) = (\text{op}, \text{ind}, |\mathbf{w}|),$$

where  $\text{op} \in \{\text{add}, \text{del}\}$ ,  $\text{ind}$  is the index of the added (or deleted) document, and  $\mathbf{w}$  is the list of keywords in this document, and  $\text{AddDB}(w)$  is the list of documents historically added to  $w$  (*cf.* Section 3.2.2).<sup>1</sup> However, no generic definition of forward privacy was given in [SPS14].

Informally, regarding the previous section, we want that an update does not leak any information about the updated keywords. In particular, the server cannot learn that the updated document matches a keyword we previously queried. With that in mind, we can give a formal and generic definition of forward privacy, closely related to the  $\mathcal{L}_{\text{SPS}}$  leakage function. This definition covers non-atomic updates, *i.e.* batched updates where several documents or several keywords are modified. Such updates cover the insertion of a new document, or the complete deletion of a document.

**Definition 4.1** (Forward privacy). *An  $\mathcal{L}$ -adaptively-secure SSE scheme  $\Sigma$  is forward private if the update leakage function  $\mathcal{L}^{\text{Updt}}$  can be written as*

$$\mathcal{L}^{\text{Updt}}(\text{op}, \text{in}) = \mathcal{L}'(\text{op}, \{(\text{ind}_i, \mu_i)\})$$

where  $\{(\text{ind}_i, \mu_i)\}$  is the set of modified documents paired with the number  $\mu_i$  of modified keywords for the updated document  $\text{ind}_i$ , and  $\mathcal{L}'$  is a stateless function (*i.e.* does not depend on previous queries).

Defined like that, we immediately have that the SPS scheme is forward private, just from the form of its leakage function. In practice, we will use a slightly stronger definition of forward privacy, defined as follows.

**Definition 4.2** (Strong forward privacy). *A  $\mathcal{L}$ -adaptively-secure SSE scheme  $\Sigma$  is strongly forward private if the update leakage function  $\mathcal{L}^{\text{Updt}}$  outputs only the operation and the number of updates:*

$$\mathcal{L}^{\text{Updt}}(\text{op}, \text{in}) = (\text{op}, \mu).$$

where  $\mu$  is the total number of modified keywords.

---

<sup>1</sup>Note that, here,  $\mathcal{L}_{\text{SPS}}^{\text{Stp}}$  is not defined, as, in the designers' mind, the encryption of the database can be done online, using the Update protocol.

Hence, by just examining the leakage function, we cannot state that SPS is strongly forward private. Yet, in Section 4.5.6, we will see that, by studying its security proof, we can actually show that SPS is strongly forward private.

In this thesis, (almost) all the forward private schemes will share the same leakage function, that we denote  $L_{\text{FP}}$ . Also, unless stated otherwise, we will only consider schemes whose update queries are insertions or deletions of single document/keyword pairs. This covers most cases as the insertion (and the deletion) of a full document can be rewritten as iteratively adding (resp. removing) the corresponding document/keyword pairs in the database.

**Definition 4.3.** *We define the leakage function  $L_{\text{FP}}$  as follows:*

$$\begin{aligned} L_{\text{FP}}^{\text{Stp}}(\text{DB}) &= N, \\ L_{\text{FP}}^{\text{Srch}}(w) &= (\text{sp}(w), \text{Hist}(w)), \\ \text{and } L_{\text{FP}}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \text{op}, \end{aligned}$$

where  $(w, \text{ind})$  is the modified pair. In particular, if the scheme supports batch updates on several keyword/document pairs, it just leaks the number of updated pairs.

### 4.3 Constraints Induced by the Forward Privacy

Unfortunately, forward privacy has various downsides in terms of efficiency. This section explains that dynamic SSE schemes have to make a tradeoff between storage efficiency, locality, and security.

#### 4.3.1 Constraints on Storage

First, let us focus on storage. A desirable property of dynamic databases is space reclamation upon entry deletion. For searchable encryption, it implies that, when a document/keyword pair is removed from the encrypted index, the logical location of the pair can be marked as empty. Now, suppose an adversary successively submits a search query for a keyword  $w$ , and then a delete query for pair  $(\text{ind}, w)$ . If the Search protocol does not modify the encrypted database beyond the entries matching the searched keyword (almost all existing schemes do not), and if the scheme reclaims space, the adversary will learn that the updated keyword was just searched. Indeed, the deletion algorithm will have to mark an entry that was just searched to be deleted, and if the search algorithm has not obviously re-located this entry, by observing the access pattern of the deletion protocol, the adversary will see that the same memory location has been accessed twice. The previous example actually shows that the locations of the encrypted keyword/document pairs before and after the search query must be completely unrelated for forward private, space-reclaiming scheme, yielding constructions with security properties close to the ones of resizable Oblivious RAM [MMBC15]. In particular, one cannot really hope for the existence of a really efficient scheme with these properties.

#### 4.3.2 Constraints on Locality

As explained in Section 3.3, Cash and Tessaro [CT14] studied the problem of the locality of memory accesses in SSE. In particular, they showed that one cannot achieve constant locality (a constant number of memory accesses), with optimal read efficiency (the server reads a constant number of bits per matching entry) without increasing the size of the encrypted database beyond  $\Theta(N)$ . Their lower bound holds for static schemes, and so applies for dynamic schemes too. Demertzis and Papamanthou described in [DP17] how to build dynamic schemes with improved locality.



Here we claim that, for dynamic schemes, locality, update efficiency and forward-privacy are irreconcilable notions. First, it is worth noticing that caring about memory locality makes sense only if the scheme is already efficient in terms of disk accesses. For example, if a scheme systematically updates the encrypted database every time a search is performed (e.g. the ORAM-based scheme [GMP16]), the rewriting cost will be much higher than the efficiency improvements due to increased locality. So without loss of generality, we consider schemes whose Search protocol does not modify a large portion of EDB. In this case, forward privacy implies that, for an updated keyword  $w$ , the location of the newly inserted tokens is unrelated to the location of already existing tokens matching  $w$ . So, if the whole set  $DB(w)$  is not somehow, at least partially, re-written during the update, no locality optimization is possible for keyword  $w$ . Said otherwise, if you want both forward privacy and locality during search, you have to do some rather large modifications of the encrypted database, either during searches or updates.

It is interesting to notice that a lower bound on the locality of ORAM was shown by Asharov *et al.* in [ACN+17]. This lower bound states that a secure  $N$ -blocks ORAM, with  $\mathcal{O}(\text{poly}(\log N))$  locality incurs  $\Omega(N^{1-\varepsilon})$  bandwidth overhead, for any constant  $\varepsilon > 0$ , even if the server stores  $\mathcal{O}(N^2)$  memory, and the client has  $\mathcal{O}(\text{poly}(\log N))$  memory.

A similar lower bound for forward private SSE is to be expected. Yet, forward private SSE has some important differences with ORAM, which can be used to construct a scheme more efficient than this ORAM lower bound. First, forward privacy does not prevent the schemes from leaking the search pattern, *i.e.* the which queries involved the same keyword  $w$ . Also, SSE almost always leaks the size pattern, and leaking this information has been shown by Asharov *et al.* to be very useful to overcome the lower bound, even in an ORAM-like setting (*cf.* the Range-ORAM and File-ORAM constructions in [ACN+17]).

In the end, even achieving logarithmic locality for a forward private scheme seems very costly, in particular in terms of bandwidth overhead during update operations. In Section 4.4, we will see how to build generically forward private schemes from static schemes, using  $\log N$  levels of exponentially increasing size, each level being a static SSE. In particular, if the static SSE construction has  $\mathcal{O}(f(N))$  locality and  $\mathcal{O}(g(N))$  read efficiency, the final scheme will have  $\mathcal{O}(f(N) \cdot \log N)$  locality and  $\mathcal{O}(g(N))$  read efficiency. For example, using the most local SSE construction of Asharov *et al.* [ANSS16] as the underlying static scheme, we end up with a forward private scheme with  $\mathcal{O}(\log N)$  locality and  $\mathcal{O}(\log N \log \log N)$  read efficiency. This construction generalizes the dynamic scheme of [DP17].

### 4.3.3 Constraints on Efficiency

Maybe the biggest constraint incurred by forward private SSE schemes is the constraint on their efficiency. Namely, the same way we showed a lower bound on the computational complexity of a scheme hiding the search pattern in Section 3.2.3 (Theorem 3.2), we can show a lower bound for forward private SSE schemes. Here, for simplicity, we suppose that the scheme only supports atomic updates, *i.e.* updates on single-keyword/document pairs. Note that we can do that without loss of generality as more general updates can be expressed as a sequence of atomic updates.

**Theorem 4.1** (Lower bound on the computational complexity of forward private SSE schemes). *Let  $\Sigma$  be a forward private SSE scheme supporting insertion and deletion of entries in the database. In particular, the leakage function leaks only  $\mathcal{L}'(\text{op}, \text{ind})$ , where  $\mathcal{L}'$  is a stateless function.  $\Sigma$  has a client state of size  $|\sigma|$ . Then, the computational complexity of the Update protocol (when summing the*

contributions of the client and of the server) is

$$\Omega\left(\frac{\log K}{\log |\sigma| \cdot \log \log K}\right).$$

*Proof.* We proceed exactly as for the proof of Theorem 3.2, with a player, able to hold at most  $b$  entries (atomic document/keyword pairs), making accesses to the encrypted database in order to answer a sequence of  $t$  update queries, and an observer who sees these accesses. As we supposed that the updates are atomic (*i.e.* are insertions or deletions of single keyword/document pairs), the sequence of  $t$  updates can be written as  $[(\text{op}_1, w_1, \text{ind}_1), \dots, (\text{op}_t, w_t, \text{ind}_t)]$ . The observer will win the game if he is able to distinguish the execution of this update sequence with the execution of the update sequence  $[(\text{op}'_1, w'_1, \text{ind}'_1), \dots, (\text{op}'_t, w'_t, \text{ind}'_t)]$  such that, for all  $1 \leq i \leq t$ ,

$$\mathcal{L}'(\text{op}_i, \text{ind}_i) = \mathcal{L}'(\text{op}'_i, \text{ind}'_i).$$

Note that if  $(\text{op}_i, \text{ind}_i) = (\text{op}'_i, \text{ind}'_i)$  for  $1 \leq i \leq t$ , then the condition clearly holds.

To answer these  $t$  queries, the player will make a sequence of  $q$  *visible* accesses  $V = (v_1, \dots, v_q)$ , observable by the adversary, and a sequence of  $(h_1, \dots, h_q)$  of *hidden actions*  $h_i$ , which the observer cannot see. As before, the player can do one of the  $b + 2$  actions among taking an entry from the cell, placing an entry in the cell or doing nothing.

We can cut this action sequence  $(v_1, h_1), \dots, (v_q, h_q)$  into  $t$  segments such that the actions  $(v_{j_{i-1}+1}, h_{j_{i-1}+1}), \dots, (v_{j_i}, h_{j_i})$  were performed during the execution of the Update protocol on input  $(\text{op}_i, w_i, \text{ind}_i)$ , with  $1 = j_0 \leq j_1 \leq \dots \leq j_t = q$ . As in the case of Lemma 3.3, the action sequence can answer at most  $b^q(b+2)^q$  update queries. On the other end, the number of update sequences with the same leakage as  $[(\text{op}_1, w_1, \text{ind}_1), \dots, (\text{op}_t, w_t, \text{ind}_t)]$  is at least  $K^t$ , as any sequence  $[(\text{op}_1, w'_1, \text{ind}_1), \dots, (\text{op}_t, w'_t, \text{ind}_t)]$  with  $w'_1, \dots, w'_t \in K$  have the same leakage (*cf.* the remark above).

As a consequence, we must have that

$$b^q(b+2)^q \geq K^t \Leftrightarrow q \geq \frac{t \log K}{\log b(b+2)}.$$

Also, we know that the computational complexity of an Update query has to be larger than  $b - |\sigma|$  (every memory cell used during the execution of the protocol has to be used unless it is useless). In the end, we have that the average complexity of an Update query is

$$\Omega\left(\max\left\{b - |\sigma|, \frac{\log K}{\log b(b+2)}\right\}\right).$$

Finally, by directly applying Lemma 3.4, we have that the average complexity of an update query is

$$\Omega\left(\frac{\log K}{\log |\sigma| \cdot \log \log K}\right).$$

□

Similarly to the lower bound of Theorem 3.2, this lower bound can be improved to

$$\Omega\left(\frac{\log K}{\log |\sigma| \cdot \log^{(a)} K}\right)$$

for any  $a \in \mathbb{N}^*$  if  $K$  is large enough, which leads us to believe that the actual lower bound is actually

$$\Omega\left(\frac{\log K}{\log |\sigma|}\right).$$

We will see in Section 4.5 that this last lower bound is tight.

## 4.4 Building a Forward Private SSE Scheme from Static Schemes

In this section, we describe a generic way to construct a forward private scheme from a static scheme. This method was used to construct the first forward private scheme, SPS, in [SPS14], and is borrowed from the hierarchical ORAM literature (*cf.* [GM11; GMOT12]). Note that the generic construction described hereunder only supports additions, but that it can easily be modified to also support deletions.

The idea, is that, to store  $N$  entries, one would use  $L = \lceil \log N \rceil$  levels  $\text{EDB}_0, \dots, \text{EDB}_{L-1}$ , where each  $\text{EDB}_i$  is a database encrypted using a static SSE scheme  $\Sigma$  storing  $2^i$  entries. A level can be empty, *i.e.* be uninitialized and set to  $\emptyset$ . The client stores the keys of the  $L$  instances, and to answer a search queries, runs the search algorithm on each of the levels, requiring at least  $\mathcal{O}(\alpha_w + \log N)$  work. Updates are more involved and work as follows. When a new atomic entry  $(w, \text{ind})$  has to be inserted, the client checks if the first level  $\text{EDB}_0$  is empty or not. If it is empty, it sets  $\text{EDB}_0$  to be the encrypted database generated by running  $\Sigma.\text{Setup}$  on the database containing only the entry  $(w, \text{ind})$ . Otherwise, the client gets the lowest empty level  $\text{EDB}_i$ , fetches the content of every level  $\text{EDB}_j$  with  $0 \leq j < i$ , resets these levels (*i.e.* sets them to the empty state), and sets  $\text{EDB}_i$  to the output of  $\Sigma.\text{Setup}(\text{DB}_{0 \rightarrow i-1})$ , where  $\text{DB}_{0 \rightarrow i-1}$  is the database consisting of the entries previously stored in levels 0 to  $i-1$ . A detailed description of this construction is given in Algorithm 4.1.

One important downside of this construction is that it requires large transient client storage during updates ( $\mathcal{O}(N)$  in the worst case) and has very large worst case computational and communication complexity: an update requires  $C_{\Sigma}^{\text{Stp}}(2^i)$  operations, where  $i$  the lowest empty level at the moment of the update, and  $C_{\Sigma}^{\text{Stp}}(m)$  is the running time of  $\Sigma$ 's Setup algorithm on input a database of size  $m$ . In particular, it is  $\Omega(N)$  when rebuilding the highest level. Yet, the amortized complexity is only  $\Omega\left(\log N \cdot \frac{C_{\Sigma}^{\text{Stp}}(N)}{N}\right)$ , and using de-amortization techniques, we can actually ensure that the average case update complexity becomes the worst-case complexity (*cf.* [GMOT12]).

One interesting feature of this generic construction is that, in some sense, the forward private scheme inherits properties from the underlying static scheme. This is particularly useful for locality and read efficiency: if the static scheme has  $\text{loc}(m)$  locality and  $\text{rdef}(m)$  for a database of size  $m$ , our generic construction will have  $\mathcal{O}(\log N \cdot \text{loc}(N))$  locality and  $\mathcal{O}(\text{reff}(N))$ . If, for example,  $\Sigma$  is Scheme I from [ANSS16], with perfect locality, optimal server storage and  $\mathcal{O}(\log N \log \log N)$  read efficiency, we end up with a forward private scheme with  $\mathcal{O}(\log N)$  locality,  $\mathcal{O}(\log N \log \log N)$  read efficiency and optimal server storage. Moreover, as the setup complexity of the scheme is  $\mathcal{O}(N)$ , the average update complexity of the induced forward private construction is  $\mathcal{O}(\log N)$ .

In [SPS14], Stefanov *et al.* use this generic idea, and add some structure to the way the elements are stored in the static schemes. This allows them to improve the efficiency of the search protocol in the presence of deletions to a  $\mathcal{O}(n_w \log^3 N)$  complexity (instead of the  $\Omega(\alpha_w + \log N)$  complexity of the generic approach). This additional structure requires to sort the levels while being rebuilt: the (re)construction of a level of size  $m$  requires  $\mathcal{O}(m \log m)$  work instead of  $\mathcal{O}(m)$ . As a consequence, the update complexity, both computational and communication, make the scheme barely practical:

---

**Algorithm 4.1** Forward privacy from static SSE.  $\Sigma$  is a static SSE scheme.

---

Setup(DB)

```

1:  $L \leftarrow \lceil \log |\text{DB}| \rceil$ 
2: Cut DB in  $\text{DB}_0, \dots, \text{DB}_{L-1}$  such that
    $|\text{DB}_i| = 2^i$  or 0 and  $\sum_{i=0}^{L-1} |\text{DB}_i| = |\text{DB}|$ 
3: for  $i = 0$  to  $L - 1$  do
4:   if  $|\text{DB}_i| \neq 0$  then
5:      $(\text{EDB}_i, K_i, \sigma_i) \leftarrow \Sigma.\text{Setup}(\text{DB}_i)$ 
6:   else  $\triangleright |\text{DB}_i| = 0$ 
7:      $(\text{EDB}_i, K_i, \sigma_i) \leftarrow (\emptyset, \emptyset, \emptyset)$ 
8:   end if
9: end for
10: return  $\left( (\text{EDB}_i)_{i=0}^{L-1}, (K_i)_{i=0}^{L-1}, (\sigma_i)_{i=0}^{L-1} \right)$ 
```

Search( $K_\Sigma, \sigma, w; \text{EDB}$ )

```

1: for  $i = 0$  to  $L - 1$  do
2:   if  $K_i \neq \emptyset$  then
3:     Run  $\Sigma.\text{Search}(K_i, \sigma_i, w; \text{EDB}_i)$ 
4:   end if
5: end for
```

Update( $K_\Sigma, \sigma, \text{op}, \text{in}; \text{EDB}$ )

```

1: Let  $\ell$  be the first empty level ( $\forall i < \ell, \text{EDB}_i \neq \emptyset$ ).
2: The client downloads and decrypts the entries in  $\text{EDB}_0, \dots, \text{EDB}_{\ell-1}$ , and places them in the
   dataset  $\text{DB}_{0 \rightarrow \ell-1}$ .
3: The client sets  $(K_i, \sigma_i) \leftarrow (\emptyset, \emptyset)$  for  $i < \ell$ .
4: The server sets  $\text{EDB}_i \leftarrow \emptyset$  for  $i < \ell$ .
5: The client runs  $(\text{EDB}_\ell, K_\ell, \sigma_\ell) \leftarrow \Sigma.\text{Setup}(\text{DB}_{0 \rightarrow \ell-1} \cup \{(w, \text{ind})\})$  and sends  $\text{EDB}_\ell$  to the
   server.
```

---

the amortized update complexity is  $\mathcal{O}(\log^2 N)$  and updates need up to 5 kB of data to be transferred for each updated entry.

Another mean of designing forward private schemes is necessary. In particular, we want something whose search and update complexity are close to the non-forward-private dynamic schemes (cf. Table 4.1). This is what we will do in Sections 4.5 and 4.6.

## 4.5 $\Sigma\text{o}\phi\text{o}\varsigma$ : Simple Optimal Forward Secure Searchable Encryption

In this section, we start by constructing a forward secure SSE scheme, whose Search and Update protocols are performed in a single round-trip, but at the cost of  $\mathcal{O}(W(\log D + \lambda))$  storage on the client side. We also first consider a scheme that only supports additions, not deletions. We will then describe how to turn this basic construction into an SSE construction supporting both additions and deletions, with reduced client storage. This construction, called  $\Sigma\text{o}\phi\text{o}\varsigma$  will be a simple optimal (for computations and communications, during both searches and updates), forward private SSE scheme<sup>2</sup>.

### 4.5.1 General Ideas

In an inverted index scheme (such as [CGKO06] and derived works), we are usually considering for each keyword  $w$  an indexed list of matching documents  $(\text{ind}_0, \dots, \text{ind}_{n_w})$ . Every element  $\text{ind}_c$

---

<sup>2</sup> $\Sigma\text{o}\phi\text{o}\varsigma$  (pronounce ‘sophos’) stands for Scalable Optimal Forward Secure Searchable Encryption. It also refers to the ancient Greek for ‘wise’: we strongly believe that it is wise to use forward private SSE.

**Table 4.1** – Comparison of dynamic SSE schemes. In all works except [SPS14], deletions are not optimally supported: the search is not linear in the number of matching documents, but in the number of inserted documents matching the query. We omitted the polynomial dependency in the security parameter  $\lambda$  for both computation and communication complexity. The  $\tilde{\mathcal{O}}(\cdot)$  notation hides the  $\log \log N$  factors. FP stands for forward privacy. Update complexities are given per updated document/keyword pair. We only considered schemes whose server’s storage complexity is optimal ( $\mathcal{O}(N)$ ). Remember that  $K$  is the number of distinct keywords in the database. The client’s storage is given as the number of entries stored by the client. As such, each entry is of size  $\log D_{\max}$ , where  $D_{\max}$  is the maximum number of documents supported by the scheme. We recall that  $n_w$  is the number of results of the query, and  $a_w$  is the number of entries historically added to  $w$  (cf. Section 3.2.2) For SPS, we give two asymptotics for the Search algorithm, as its efficiency depends on the algorithm chosen to execute the query.

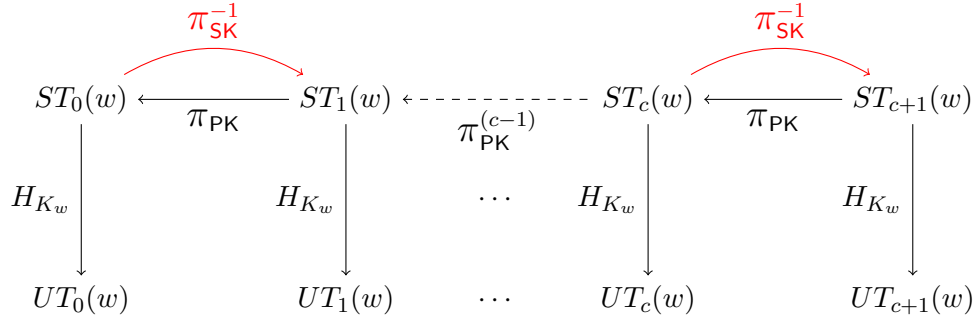
Scheme	Computation		Communication		Client Storage	FP
	Search	Update	Search	Update		
$\Pi^{\text{dyn}}$ [CJJ+14]	$\mathcal{O}(a_w)$	$\mathcal{O}(1)$	$\mathcal{O}(n_w)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	✗
SPS [SPS14]	$\mathcal{O}(a_w + \log N)$ $\mathcal{O}(n_w \log^3 N)$	$\mathcal{O}(\log^2 N)$	$\mathcal{O}(n_w + \log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(N^\alpha)$	✓
TWORAM [GMP16]	$\tilde{\mathcal{O}}\left(\begin{matrix} a_w \log N \\ + \log^3 N \end{matrix}\right)$	$\tilde{\mathcal{O}}(\log^2 N)$	$\tilde{\mathcal{O}}\left(\begin{matrix} a_w \log N \\ + \log^3 N \end{matrix}\right)$	$\tilde{\mathcal{O}}(\log^3 N)$	$\mathcal{O}(1)$	✓
$\Sigma\phi\phi\sigma$ § 4.5	$\mathcal{O}(a_w)$	$\mathcal{O}(1)$	$\mathcal{O}(n_w)$	$\mathcal{O}(1)$	$\mathcal{O}(K)$	✓
Diana § 4.6	$\mathcal{O}(a_w)$	$\mathcal{O}(\log a_w)$	$\mathcal{O}(n_w + \log a_w)$	$\mathcal{O}(1)$	$\mathcal{O}(K)$	✓
Lower bound Theorem 4.1	$\Omega(n_w)$	$\Omega\left(\frac{\log K}{\log  \sigma  \log \log K}\right)$	$\Omega(n_w)$	$\Omega(1)$	$\Theta( \sigma )$	✓

of this list is then encrypted and stored at a (logical) location derived from  $w$  and  $c$ . We call this location  $UT_c(w)$ . When the client wants to add a document matching  $w$ , he computes a new location  $UT_{n_w+1}(w)$ , encrypts the document index as  $e$ , and sends  $(UT_{n_w+1}(w), e)$  to the server (this explains our notation  $UT$ , for update token).

When the client performs a search query on  $w$ , he will issue a search token that will allow the server to recompute the update tokens, and hence the locations of the entries matching  $w$ . In general, we want the update tokens for a given  $w$  to be unlinkable until a search token  $ST(w)$  is issued. In our case, the search token generated by the client will depend on the number  $n_w$  of matching entries, and we want that the search token  $ST_c(w)$  generated for  $c$  results to be unlinkable to the update tokens  $UT_i(w)$  for  $i > c$ , i.e. the update tokens that will be issued for keyword  $w$  in the future. In particular, it implies that the server cannot compute  $ST_i(w)$  from  $ST_c(w)$  when  $i > c$ .

To do so, we could make the client generate all the search tokens using a PRF evaluation  $F(w, i)$ , and send them to the server. However this solution is not satisfactory: the client needs to send  $\mathcal{O}(n_w)$  tokens to the server, which can be a problem on constrained devices. Here, we propose another solution, based on trapdoor permutations: from  $ST_i(w)$ , the server will be able to compute  $ST_{i-1}(w)$  using the a public key, but only the client will be able to construct  $ST_{i+1}(w)$ .

Figure 4.1 gives the relations among the tokens and formalizes our idea for token generation.



**Figure 4.1** – Relations among search and update tokens. Operations in red can only be done by the client, using the secret key SK.

$ST_{i+1}(w)$  is generated from  $ST_i(w)$  by applying the inverse of a one-way trapdoor permutation  $\pi$ : only the client will be able to perform this operation, while the server, given the public key PK will do the opposite, namely compute from a search token  $ST_c(w)$  all the tokens  $ST_i(w)$  for  $0 \leq i < c$ . Finally, the update tokens are derived from the search tokens, using a keyed hash function. In particular, it is crucial that  $H$  is pre-image resistant for the security of the scheme. We will actually show the security of this construction when  $H$  is modeled as a random oracle.

#### 4.5.2 Basic Construction

Algorithm 4.2 gives the formal description of our basic forward private scheme,  $\Sigma o\phi o\varsigma$ -B. It follows the idea of the previous section for the token generation. Also, the only type of updates  $\Sigma o\phi o\varsigma$ -B supports is the insertion of new keyword/document pairs.

In the pseudo code,  $\pi$  is a trapdoor permutation,  $F$  is a PRF,  $H_1$  and  $H_2$  are keyed hash functions, whose outputs are, respectively,  $\mu$  and  $\ell$  bits long. On the client side,  $\mathbf{W}$  maps every inserted keyword to its current search token  $ST_c(w)$  and to a counter  $c = n_w - 1$ . Every time a new document matching  $w$  is inserted,  $\mathbf{W}[w]$  gets ‘incremented’: the client generates the new search token  $ST_{c+1}(w) = \pi^{-1}(ST_c(w))$  and stores it in  $\mathbf{W}$ . If  $w$  did not match any documents, a new  $ST_0(w)$  is randomly picked and put in  $\mathbf{W}$ . Finally, the entries’ locations, *i.e.* the update tokens, are derived from the search tokens by a keyed hash function.

$\Sigma o\phi o\varsigma$ ’ setup algorithm does not take a database as input: as stated in Section 4.1, encryption can be performed online with a forward private scheme, without loss of security.

**Correctness.** The correctness of  $\Sigma o\phi o\varsigma$ -B is quite straightforward. The only issue is collision among the update tokens  $UT_c(w)$ , generated from  $H_1$  with input  $(K_w, ST_c)$ . We can reduce the correctness to the collision resistance of  $H_1$ . In particular, we need to choose  $\mu$  such that  $N^2/2^\mu$  is negligible in the security parameter, so in practice, we will set  $\mu = \lambda + 2 \log N_{\max}$ , where  $N_{\max}$  is the maximum number of pairs the database can store.

**Complexity.** The scheme’s computational complexity is optimal:  $\mathcal{O}(n_w)$  for a search query,  $\mathcal{O}(1)$  for an update. Both Search and Update are single round, and their performance will not be more affected by network latency than regular insecure protocols.

Bandwidth is also (almost) optimal. The Search protocol uses a token that is a single element in  $\mathcal{M}$  (the domain of the trapdoor permutation), resulting in  $\log |\mathcal{M}| = \text{poly}(\lambda)$  bits token per query.

---

**Algorithm 4.2**  $\Sigma\phi\phi\phi\phi$ -B: Forward private SSE scheme with large client storage.
 

---

Setup()

- 1:  $K_S \xleftarrow{\$} \{0, 1\}^\lambda$
- 2:  $(SK, PK) \leftarrow \pi.\text{KeyGen}(1^\lambda)$
- 3:  $\mathbf{W}, \mathbf{T} \leftarrow \text{empty map}$
- 4: **return**  $((\mathbf{T}, PK), (K_S, SK), \mathbf{W})$

Search( $K_\Sigma, w, \sigma; \text{EDB}$ )*Client:*

- 1:  $K_w \leftarrow F_{K_S}(w)$
- 2:  $(ST_c, c) \leftarrow \mathbf{W}[w] \quad \triangleright c = n_w - 1$
- 3: **if**  $(ST_c, c) = \perp$
- 4:   **return**  $\emptyset$
- 5: Send  $(K_w, ST_c, c)$  to the server.

*Server:*

- 6: **for**  $i = c$  **to** 0 **do**
- 7:    $UT_i \leftarrow H_1(K_w, ST_i)$
- 8:    $e \leftarrow \mathbf{T}[UT_i]$
- 9:    $\text{ind} \leftarrow e \oplus H_2(K_w, ST_i)$
- 10:   Output each ind
- 11:    $ST_{i-1} \leftarrow \pi_{PK}(ST_i)$
- 12: **end for**

Update( $K_\Sigma, \text{add}, w, \text{ind}, \sigma; \text{EDB}$ )*Client:*

- 1:  $K_w \leftarrow F(K_S, w)$
- 2:  $(ST_c, c) \leftarrow \mathbf{W}[w]$
- 3: **if**  $(ST_c, c) = \perp$  **then**
- 4:    $ST_0 \xleftarrow{\$} \mathcal{M}, c \leftarrow -1$
- 5: **else**
- 6:    $ST_{c+1} \leftarrow \pi_{SK}^{-1}(ST_c)$
- 7: **end if**
- 8:  $\mathbf{W}[w] \leftarrow (ST_{c+1}, c + 1)$
- 9:  $UT_{c+1} \leftarrow H_1(K_w, ST_{c+1})$
- 10:  $e \leftarrow \text{ind} \oplus H_2(K_w, ST_{c+1})$
- 11: Send  $(UT_{c+1}, e)$  to the server.

*Server:*

- 12:  $\mathbf{T}[UT_{c+1}] \leftarrow e$
- 

The Update protocol sends a  $\mu + \ell$  bits token per updated document/keyword pair, representing a  $\lambda + \log N_{\max}$  bits increase compared to the smallest possible update token size of an unencrypted database.

The client's storage is  $\mathcal{O}(K(\log |\mathcal{M}| + \log D))$ : an element of  $\mathcal{M}$  is stored for every keyword, with the counter  $c < D$ .

### 4.5.3 Security

The adaptive security of  $\Sigma\phi\phi\phi\phi$ -B can be proven in the Random Oracle Model, and relies on the one-wayness of the TDP  $\pi$  and on the pseudo-randomness of  $F$ . We start by giving a sketch of the proof, and then full proof.

**Theorem 4.2** (Adaptive security of  $\Sigma\phi\phi\phi\phi$ -B). *Let  $\pi$  be a one-way trapdoor permutation,  $F$  a PRF, and  $H_1$  and  $H_2$  two hash functions modeled as a random oracle outputting respectively  $\mu$  and  $\lambda$  bits. We recall from Definition 4.3 that the leakage function  $L_{\text{FP}}$  is defined as*

$$L_{\text{FP}}^{\text{Srch}}(w) = (\text{sp}(w), \text{Hist}(w))$$

$$L_{\text{FP}}^{\text{Updt}}(\text{add}, w, \text{ind}) = \perp.$$

Then  $\Sigma\phi\phi\phi\phi$ -B is  $L_{\text{FP}}$ -adaptively-secure. More precisely, for any polynomial-time adversary  $A$  encrypting a database of at most  $N$  entries, with at most  $K$  distinct keywords, there exists polynomial-time adversaries  $B_1$  and  $B_2$ , and a simulator  $S$  such that

$$\text{Adv}_{\Sigma, S, L_{\text{FP}}, A}^{\text{SSE-sim}}(\lambda) \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda) + 2N \cdot \text{Adv}_{\pi, B_2}^{\text{tdp}}(\lambda)$$

where  $B_1$  makes at most  $K$  queries.

Hence,  $\Sigma\phi\phi\sigma$  thwarts all the devastating adaptive file-injection attacks of [ZKP16], but not the non-adaptive ones.

The proof works by constructing successive indistinguishable hybrids, where  $H_1$  and  $H_2$  are modeled as random oracles, the first hybrid being the real-world game, and the last the ideal-world game. We first replace  $F$  by a random function, *i.e.* we randomly pick the keys  $K_w$ .

In a second step, we will replace all the strings generated by the random oracles in the Update protocol by randomly chosen strings. The game will then program the random oracles during the Search protocol so that the result produced by the server matches the real results:  $H_1$  is set to map the  $i$ -th search token for  $w$  to the update token produced randomly when  $w$  was updated for the  $i$ -th time.  $H_2$  is programmed in a similar manner to produce the right keystream used to hide the ind values. We show that if the first and second hybrids are *not* indistinguishable, it means that the adversary was able to invert the trapdoor permutation without knowing the secret key.

Finally, we construct a hybrid that only needs to know the repetition of search queries and the history to produce search tokens indistinguishable from the previous hybrid. As this hybrid only needs the output of the leakage function to run, it means that we have a simulator that produces indistinguishable transcripts from the real security game.

Note that  $L_{FP}$  uses Hist and not DB because the simulator needs to know exactly when documents matching  $w$  were inserted in the database in order to correctly simulate the real protocol.

*Proof.* We are going to derive several games from the real world game  $\text{SSEReAL}_{\Sigma\phi\phi\sigma}^A(1^\lambda)$ .

**Game  $G_0$ .**  $G_0$  is exactly the real world SSE security game  $\text{SSEReAL}$ .

$$\mathbb{P}[\text{SSEReAL}_{\Sigma\phi\phi\sigma}^A(1^\lambda) = 1] = \mathbb{P}[G_0^A(1^\lambda) = 1].$$

**Game  $G_1$ .** Instead of calling  $F$  when generating  $K_w$ ,  $G_1$  picks a new random key when it is confronted to a new  $w$ , and stores it in a table Key so it can be reused next time  $w$  is queried. If an adversary is able to distinguish between  $G_0$  and  $G_1$ , we can then build a reduction able to distinguish between  $F$  and a truly random function. More formally, there exists an efficient adversary  $B_1$ , making at most  $K$  oracle queries, such that

$$\mathbb{P}[G_0^A(1^\lambda) = 1] - \mathbb{P}[G_1^A(1^\lambda) = 1] \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda).$$

**Game  $G_2$ .** In  $G_2$ , in the Update protocol, instead of calling  $H_1$  to generate the update tokens  $UT$ , we pick random strings. Then, during the Search protocol, the random oracle  $H_1$  is programmed so that  $H_1(K_w, ST_c(w)) = UT_c(w)$ .

Algorithm 4.3 formally describes  $G_2$ , and also introduces an intermediate game  $\widetilde{G}_2$ , by including the additional boxed lines. In the pseudo-code, we explicit the calls to the random oracle  $H_1$ , and keep track of the transcripts *via* the table  $H_1$ . In particular, we can see that we explicitly program the RO during Search at line 7. Note that we use the following convention for the table Key: if an entry is accessed for the first time, it is first randomly chosen and then returned. Also  $G_2$  and  $\widetilde{G}_2$  make some bookkeeping of the search tokens  $ST_c$ , instead of recomputing all of them in Search.

The point of  $\widetilde{G}_2$  is to ensure consistency of  $H_1$ 's transcript: in  $\widetilde{G}_2$ ,  $H_1$  is never programmed to two different values for the same input by Search' line 7. Instead of immediately generating the  $UT$  derived from the  $c$ -th  $ST$  for keyword  $w$  from  $H_1$ ,  $\widetilde{G}_2$  randomly either chooses them if  $(K_w, ST_{c+1})$  does not already appear in  $H_1$ 's transcript, or, if this is already the case, sets  $UT_{c+1}$  to the already chosen value  $H_1[K_w, UT_{c+1}]$ . Then,  $\widetilde{G}_2$  lazily programs the RO when needed by the Search protocol



---

**Algorithm 4.3** Games  $G_2$  and  $\widetilde{G}_2$  Boxed code is included in  $\widetilde{G}_2$  only.

---

Setup()	$Update(K_\Sigma, \text{add}, w, \text{ind}, \sigma; \text{EDB})$
1: $(\text{SK}, \text{PK}) \leftarrow \text{KeyGen}(1^\lambda)$ 2: $\mathbf{W}, \mathbf{T} \leftarrow \text{empty map}$ 3: $\text{bad} \leftarrow \text{false}$ 4: <b>return</b> $((\mathbf{T}, \text{PK}), (K_S, \text{SK}), \mathbf{W})$	<i>Client:</i> 1: $K_w \leftarrow \text{Key}[w]$ 2: $(ST_0, \dots, ST_c, c) \leftarrow \mathbf{W}[w]$ 3: <b>if</b> $(ST_0, \dots, ST_c, c) = \perp$ <b>then</b> 4: $ST_0 \xleftarrow{\$} \mathcal{M}, c \leftarrow -1$ 5: <b>else</b> 6: $ST_{c+1} \leftarrow \pi_{\text{SK}}^{-1}(ST_c)$ 7: <b>end if</b> 8: $\mathbf{W}[w] \leftarrow (ST_0, \dots, ST_{c+1}, c+1)$ 9: $UT_{c+1} \leftarrow \{0, 1\}^\mu$ 10: <b>if</b> $H_1(K_w, ST_{c+1}) \neq \perp$ <b>then</b> 11: <span style="border: 1px solid black; padding: 2px;"><math>\text{bad} \leftarrow \text{true}</math></span> 12: <span style="border: 1px solid black; padding: 2px;"><math>UT_{c+1} \leftarrow H_1(K_w, ST_{c+1})</math></span> 13: <b>end if</b> 14: $UT[w, c+1] \leftarrow UT_{c+1}$ 15: $d \leftarrow \text{ind} \oplus H_2(K_w, ST_{c+1})$ 16: <b>Send</b> $(UT_{c+1}, d)$ to the server. <i>Server:</i> 17: $\mathbf{T}[UT_{c+1}] \leftarrow e$
Search( $K_\Sigma, w, \sigma; \text{EDB}$ ) <i>Client:</i> 1: $K_w \leftarrow \text{Key}[w]$ 2: $(ST_0, \dots, ST_c, c) \leftarrow \mathbf{W}[w]$ 3: <b>if</b> $(ST_0, \dots, ST_c, c) = \perp$ 4: <b>return</b> $\emptyset$ 5: $(\text{ind}_0, \dots, \text{ind}_c) \leftarrow \text{AddDB}(w) \triangleright \text{Ordered}$ <i>from the order of updates</i> 6: <b>for</b> $i = 0$ <b>to</b> $c$ <b>do</b> 7: $H_1(K_w, ST_i) \leftarrow UT[w, i]$ 8: <b>end for</b> 9: <b>Send</b> $(K_w, ST_c, c)$ to the server. <i>Server:</i> 10: <b>for</b> $i = c$ <b>to</b> $0$ <b>do</b> 11: $UT_i \leftarrow H_1(K_w, ST_i)$ 12: $e \leftarrow \mathbf{T}[UT_i]$ 13: $\text{ind} \leftarrow e \oplus H_2(K_w, ST_i)$ 14: <b>Output each</b> $\text{ind}$ 15: $ST_{i-1} \leftarrow \pi_{\text{PK}}(ST_i)$ 16: <b>end for</b>	$H_1(k, st)$ 1: $v \leftarrow H_1(k, st)$ 2: <b>if</b> $v = \perp$ <b>then</b> 3: $v \xleftarrow{\$} \{0, 1\}^\lambda$ 4: <b>if</b> $\exists w, c$ s.t. $st = ST_c \in \mathbf{W}[w]$ <b>then</b> 5: <span style="border: 1px solid black; padding: 2px;"><math>\text{bad} \leftarrow \text{true}, v \leftarrow UT[w, c]</math></span> 6: <b>end if</b> 7: $H_1(k, st) \leftarrow v$ 8: <b>end if</b> 9: <b>return</b> $v$

---

(line 7) or by an adversary's query (line 5 of  $H_1$ ), so that its outputs are consistent with the chosen values of the  $UT$ 's.

Because of this,  $H_1$ 's outputs in  $\widetilde{G}_2$  and  $G_1$  are perfectly indistinguishable, and so are the games:

$$\mathbb{P}[\widetilde{G}_2^A(1^\lambda) = 1] = \mathbb{P}[G_1^A(1^\lambda) = 1].$$

The games  $\widetilde{G}_2$  and  $G_2$  are also perfectly identical unless the flag  $\text{bad}$  is set to true, and we can apply the *identical-until-bad* technique to bound the distinguishing advantage between  $\widetilde{G}_2$  and  $G_2$ :

$$\mathbb{P}[\widetilde{G}_2^A(1^\lambda) = 1] - \mathbb{P}[G_2^A(1^\lambda) = 1] \leq \mathbb{P}[\text{bad is set to true in } \widetilde{G}_2^A(1^\lambda)].$$

Intuitively, we can see that, if  $\text{bad}$  is set to true, we can break the one-wayness of the TDP. More formally, we are going to construct a reduction  $B_2$  from a distinguisher  $A$  inserting  $N$  keyword/document pairs in the database, using a technique similar to the Schnorr's signatures proofs. We note

the maximum number of documents matching a keyword  $\max_{w \in W} n_w = n_{\max}$ . We assume that  $n_{\max} = \text{poly}(\lambda)$ .  $B_2$  will take as input a public key  $\text{PK}$  and a challenge  $y \in \mathcal{M}$ , and will output  $x$  such that  $\pi_{\text{PK}}(x) = y$ .

As for Schnorr's signatures proofs,  $B_2$  first guesses the pair  $(w^*, c^*)$  for which bad will be set to true for the first time, by querying  $H_1$  on  $(K_{w^*}, ST_{c^*})$  (i.e. by pre-computing  $\text{UT}[w^*, c^*]$ ), among the  $N$  possible pairs. For all keyword  $w \in W \setminus \{w^*\}$ ,  $B_2$  pre-computes  $ST_i(w)$  as follows:

$$\begin{aligned} ST_{n_{\max}}(w) &\stackrel{\$}{\leftarrow} \mathcal{M}, \\ ST_i(w) &\leftarrow \pi_{\text{PK}}(ST_{i+1}(w)) \text{ for } 0 \leq i < n_{\max} \end{aligned}$$

Similarly, for  $w^*$ ,  $B_2$  generates the search tokens from the challenge  $y$ :

$$\begin{aligned} ST_{c^*-1}(w^*) &\leftarrow y, \\ ST_i(w^*) &\leftarrow \pi_{\text{PK}}(ST_{i+1}(w^*)) \text{ for } 0 \leq i < c^* \end{aligned}$$

It is essential to see that the distribution of the search tokens remains unchanged from game  $G_2$ . So, in order to return a pre-image of  $y$ , the reduction  $B_2$  will find the value  $x$  by evaluating  $\pi_{\text{PK}}(r)$  for all  $(K_{w^*}, r)$  in the RO's transcript and check if  $\pi_{\text{PK}}(r) = y$ . Hence, for a fixed pair  $(w^*, c^*)$ , if  $G_2$  sets bad to true because of an adversary's query on  $(K_{w^*}, ST_{c^*})$ ,  $B_2$  is able to invert  $\pi$  without the secret key:

$$\mathbb{P}[\text{bad is set to true by forging } \text{UT}[w^*, c^*]] = \text{Adv}_{\pi, B_2}^{\text{tdp}}(\lambda).$$

Guessing the pair  $(w^*, c^*)$  implies a  $N$  loss factor in the advantage of the reduction, and

$$\begin{aligned} \mathbb{P}[G_1^A(1^\lambda) = 1] - \mathbb{P}[G_2^A(1^\lambda) = 1] &= \mathbb{P}[\widetilde{G}_2^A(1^\lambda) = 1] - \mathbb{P}[G_2^A(1^\lambda) = 1] \\ &\leq N \cdot \text{Adv}_{\pi, B_2}^{\text{tdp}}(\lambda). \end{aligned}$$

**Game  $G_3$ .** Game  $G_3$  does exactly what game  $G_2$  did for  $H_1$ , but for  $H_2$ . The exact same argument can be reused, giving that there is an adversary  $B_3$  such that

$$\mathbb{P}[G_2^A(1^\lambda) = 1] - \mathbb{P}[G_3^A(1^\lambda) = 1] \leq N \cdot \text{Adv}_{\pi, B_3}^{\text{tdp}}(\lambda).$$

Note that we can consider that  $B_2 = B_3$  without loss of generality: we could have started with  $H_2$  instead of  $H_1$  and the reduction would have been the same.

**Game  $G_4$ .** In game  $G_4$ , as defined by Algorithm 4.4, the game keeps track of the randomly generated string  $UT$  and  $d$  differently than before, but the transcript output by Search and Update are strictly identical, and the random oracles are also programmed identically. In Algorithm 4.4, we removed the now useless code for the  $H_1$  oracle. Also note that we got rid of the server's part in the protocols: these are single round-trip protocols and the removed lines do not influence the client's transcript.

We still have to show that  $G_3$  and  $G_4$  are indistinguishable. For Update, this is immediate as we are already outputting fresh random strings for each update in  $G_3$ . In Search,  $G_4$  generates the search token from  $ST_0$  by iterating  $\pi_{\mathcal{SK}}^{-1}$  instead of using an already computed and stored token (note that for  $\mathbf{W}$ , we adopt the same convention than for Key: if an entry is accessed for the first time, the game randomly picks it in  $\mathcal{M}$  and stores it in  $\mathbf{W}$ ).

Finally, instead of directly mapping the pairs  $(w, i)$  (a keyword and the  $i$ -th update to this keyword) to the values picked for  $UT$  and  $e$ , we use the intermediate table  $\text{Updates}$  that maps  $(w, i)$  to the

**Algorithm 4.4** Game  $G_4$ .

Setup()	Search( $K_\Sigma, w, \sigma$ ; EDB)
1: $(\text{SK}, \text{PK}) \leftarrow \text{KeyGen}(1^\lambda)$ 2: $\mathbf{W}, \mathbf{T} \leftarrow \text{empty map}$ 3: $u \leftarrow 0$ 4: $\text{Updates} \leftarrow \text{empty map}$ 5: <b>return</b> $((\mathbf{T}, \text{PK}), (K_S, \text{SK}), \mathbf{W})$	<i>Client:</i> 1: $K_w \leftarrow \text{Key}[w]$ 2: $ST_0 \leftarrow \mathbf{W}[w]$ 3: $[(u_0, \text{ind}_0), \dots, (u_c, \text{ind}_c)] \leftarrow \text{Updates}[w]$ 4: <b>if</b> $c = 0$ 5: <b>return</b> $\emptyset$ 6: <b>for</b> $i = 0$ <b>to</b> $c$ <b>do</b> 7:   Program $H_1$ and $H_2$ : $H_1(K_w, ST_i) \leftarrow \text{UT}[u_i]$ $H_2(K_w, ST_i) \leftarrow \mathbf{e}[u_i] \oplus \text{ind}_i$ 8: $ST_{i+1} \leftarrow \pi_{\text{SK}}^{-1}(ST_i)$ 9: <b>end for</b> 10: <b>Send</b> $(K_w, ST_c, c)$ to the server.
<i>Update(<math>K_\Sigma, \text{add}, w, \text{ind}, \sigma</math>; EDB)</i> <i>Client:</i> 1: <b>Append</b> $(u, \text{ind})$ to $\text{Updates}[w]$ 2: $\text{UT}[u] \xleftarrow{\$} \{0, 1\}^\mu$ 3: $\mathbf{e}[u] \xleftarrow{\$} \{0, 1\}^\lambda$ 4: <b>Send</b> $(\text{UT}[u], \mathbf{e}[u])$ to the server. 5: $u \leftarrow u + 1$	

global update count (the map is implicit as  $\text{Updates}[w]$  stores a list of update counter values that gets appended with the current counter when  $w$  is updated). Hence,

$$\mathbb{P}[G_3^A(1^\lambda) = 1] - \mathbb{P}[G_4^A(1^\lambda) = 1] = 0.$$

**The Simulator.** We can cut the code of game  $G_4$  in two independent parts: the leakage and the simulator. The simulator is described in Algorithm 4.5, and the leakage function is  $L_{\text{FP}}$ .  $G_4$  and  $\text{SSEIDEAL}_{S, L_{\text{FP}}}$  are identical games, the only difference being that, instead of the keyword  $w$ ,  $S$  uses the counter  $\bar{w} = \min \text{sp}(w)$  uniquely mapped from  $w$  using the leakage function. Hence,

$$\mathbb{P}[G_4^A(1^\lambda) = 1] - \mathbb{P}[\text{SSEIDEAL}_{\Sigma\phi\phi\phi, S, L_{\text{FP}}}^A(1^\lambda) = 1] = 0.$$

**Conclusion.** By combining all the contributions from all the games, there exists 2 adversaries  $B_1$  and  $B_2$  such that

$$\mathbb{P}[\text{SSEReAL}_{\Sigma\phi\phi\phi}^A(1^\lambda) = 1] - \mathbb{P}[\text{SSEIDEAL}_{\Sigma\phi\phi\phi, S, L_{\text{FP}}}^A(1^\lambda) = 1] \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda) + 2N \cdot \text{Adv}_{\pi, B_2}^{\text{tdp}}(\lambda).$$

□

#### 4.5.4 Derived Constructions

**Deletion Support.** Although  $\Sigma\phi\phi\phi$ -B does not support deletions, this is easy to fix by ‘duplicating’ the data structure: we will use one instance of  $\Sigma\phi\phi\phi$ -B for insertions, and the other for deletions. When searching  $w$ , the server will compute and return the difference between the indices matching  $w$  in both instances.

The leakage stays the same: we can separate the elements of  $\text{Hist}(w)$  according to their operation (add or del) in two sublists  $\text{Hist}_{\text{add}}(w)$  and  $\text{Hist}_{\text{del}}(w)$  to build the leakage functions of each instance of  $\Sigma\phi\phi\phi$ -B. The only difference with the original scheme would be that it leaks the operation  $\text{op} = \text{add}$  or  $\text{del}$ . Yet, we can use the same map  $\mathbf{T}$  to store the entries for both instances, which would hide the actual operation performed during the update.

**Algorithm 4.5** Proof of  $\Sigma\phi\phi\phi$ : simulator  $S$ 

<u><math>S.Setup()</math></u>	<u><math>S.Search(sp(w), Hist(w))</math></u>
1: $(SK, PK) \leftarrow \text{KeyGen}(1^\lambda)$ 2: $\mathbf{W}, \mathbf{T} \leftarrow \text{empty map}$ 3: $u \leftarrow 0$ 4: <b>return</b> $(\mathbf{T}, PK)$	<i>Client:</i> 1: $\bar{w} \leftarrow \min sp(x)$ 2: $K_{\bar{w}} \leftarrow \text{Key}[\bar{w}]$ 3: $ST_0 \leftarrow \mathbf{W}[\bar{w}]$ 4: Parse $Hist(w)$ as $[(u_0, \text{add}, \text{ind}_0), \dots, (u_c, \text{add}, \text{ind}_c)]$ 5: <b>if</b> $c = 0$ 6: <b>return</b> $\emptyset$ 7: <b>for</b> $i = 0$ <b>to</b> $c$ <b>do</b> 8:   Program $H_1$ and $H_2$ : $H_1(K_w, ST_i) \leftarrow UT[u_i]$ $H_2(K_w, ST_i) \leftarrow e[u_i] \oplus \text{ind}_i$ 9: $ST_{i+1} \leftarrow \pi_{SK}^{-1}(ST_i)$ 10: <b>end for</b> 11: Send $(K_{\bar{w}}, ST_c)$ to the server.
<u><math>S.Update()</math></u> <i>Client:</i> 1: $UT[u] \xleftarrow{\$} \{0, 1\}^\mu$ 2: $e[u] \xleftarrow{\$} \{0, 1\}^\lambda$ 3: Send $(UT[u], e[u])$ to the server. 4: $u \leftarrow u + 1$	

**Batch Updates.** We can also slightly modify the update protocol to support batch updates on documents: when we want to update document with index  $\text{ind}$  on the keywords list  $\mathbf{w}$ , we successively select keywords in  $\mathbf{w}$  in random order, and run the original Update protocol with input  $\text{ind}$  and the selected keyword.

Again, the leakage function remains identical. We just have to slightly update the security proof: when the simulator programs the random oracles on  $(K_w, ST_i)$ , instead of targeting exactly the only token produced at the  $i$ -th update on  $w$ , it will pick a one of the random tokens produced during the update that modified  $DB(w)$  for the  $i$ -th time. This is the reason why picking keywords in a random order during update is important: if they were picked sequentially, the simulator would have to have access to the insertion order.

#### 4.5.5 Reducing Client-side Storage

We saw that the client's storage is  $\mathcal{O}(K(\log |\mathcal{M}| + \log D))$ . This can be a problem on constrained devices, especially when  $\mathcal{M}$  is big, which is the case for both RSA and Rabin's Squaring trapdoor permutations:  $\mathcal{M} = \mathbb{Z}_N^*$ , where  $N$  is, for a reasonable level of security, a 2048 bits integer.

But there is a workaround to reduce storage to  $\mathcal{O}(K \log D)$  at the expense of additional computations. The idea is to pseudo-randomly generate  $ST_0(w)$  from  $w$  (or a unique identifier  $i_w \in \mathbb{N}$  of  $w$ ). When  $\mathcal{M} = \mathbb{Z}_N^*$ , this is quite easy to do from a PRF  $G_{K_{\text{TDP}}} : \mathbb{N} \rightarrow \{0, 1\}^{\lambda + \log N}$  by taking  $ST_0(w) \leftarrow G_{K_{\text{TDP}}}(i_w) \bmod N$ .

When  $ST_c(w)$  is needed, we recompute it from  $ST_0(w)$ . However, this will be very computationally expensive if we have to iteratively compute  $\pi_{SK}^{(-c)}$  by iterating  $\pi_{SK}^{-1}$   $c$  times. Fortunately, this is not the case for common trapdoor permutations  $\pi$ , and in particular for RSA: if  $(p, q, d)$  and  $(N, e)$  are respectively the secret and the public keys,  $y = \text{RSA}_{SK}^{(-i)}(x)$  can be computed as follows.

$$f \leftarrow d^i \bmod (p-1)(q-1),$$

$$y \leftarrow x^f \bmod N.$$

We can also use the Chinese remainders technique to improve the practical performance of this computation.

Then, the client has only to store an identifier  $i_w$  for every  $w$ , together with the counter  $c$ . When he needs to, he can easily recompute the  $ST_c(w)$  from  $i_w$  and  $c$ , with a small complexity overhead (essentially the cost of a private-key operation, when RSA is the chosen TDP). The client's storage reduces to  $\mathcal{O}(K \log D)$  asymptotically (as  $c < D$ ).

We call this version  $\Sigma\phi\phi\phi$ -B (without the -B), and it is the one that we implemented. The proof of security of  $\Sigma\phi\phi\phi$ -B can easily be ported to  $\Sigma\phi\phi\phi$ .

**Theorem 4.3.**  $\Sigma\phi\phi\phi$  is  $L_{\text{FP}}$ -adaptively-secure.

#### 4.5.6 Comparison with Other Constructions

If we limit this scheme to only support modification of full documents (*i.e.* it is not allowed to add/delete a keyword to a document already in the database, but we can add or delete an entire document), our scheme has the same functionalities as the SPS scheme [SPS14], whose leakage function is  $\mathcal{L}_{\text{SPS}}$ , with  $\mathcal{L}_{\text{SPS}}^{\text{Srch}}(w) = (\text{sp}(w), \text{DB}(w), \text{AddDB}(w))$ , and  $\mathcal{L}_{\text{SPS}}^{\text{Updt}}(\text{op}, \text{ind}, \mathbf{w}) = (\text{op}, \text{ind}, |\mathbf{w}|)$ .  $L_{\text{FP}}$  and  $\mathcal{L}_{\text{SPS}}$  might look incomparable, but actually we can construct  $L_{\text{FP}}$  from  $\mathcal{L}_{\text{SPS}}$  rather easily: we just reconstruct  $\text{Hist}(w)$  from  $\text{AddDB}(w)$  and the timestamps  $t$  generated at every update. We associate these timestamps to the leaked information  $(\text{op}, \text{ind}, |\mathbf{w}|)$ , and for each matching document in  $\text{AddDB}(w)$ , we can retrieve the associated update operation, and when it happened, and hence recompute  $\text{Hist}(w)$ .

This shows that  $\Sigma\phi\phi\phi$  is  $\mathcal{L}_{\text{SPS}}$ -adaptively-secure. We can similarly show that SPS is  $L_{\text{FP}}$ -adaptively-secure by adapting the proof of [SPS14]. Hence, SPS and  $\Sigma\phi\phi\phi$  have exactly the same security guarantees. However  $\Sigma\phi\phi\phi$  is much more efficient in terms of bandwidth usage and update complexity. Namely, SPS's updates trigger  $\mathcal{O}(\log^2 N)$  work and  $\mathcal{O}(\log N)$  bandwidth usage per update, using standard, yet not trivial de-amortization techniques.  $\Sigma\phi\phi\phi$  bandwidth and computational overheads for an update are both a constant.

Also, SPS needs  $\mathcal{O}(N^\alpha)$  client memory for  $0 < \alpha < 1$  as working storage to run the oblivious sort algorithm needed during the updates, while  $\Sigma\phi\phi\phi$  stores  $\mathcal{O}(K \log D)$  on the client – and we expect  $K \ll N$ .

As presented in Table 4.1,  $\Sigma\phi\phi\phi$  has the same asymptotic complexity as the most efficient dynamic SSE schemes. Moreover, we saw in Section 4.3 that we cannot really hope for something better in terms of features (space reclamation) or storage locality, for a forward private scheme. Moreover,  $\Sigma\phi\phi\phi$  shows that the lower bound of Theorem 4.1 is essentially tight: as the client's state stores  $\mathcal{O}(K)$  elements of size  $\mathcal{O}(D_{\text{max}})$  (where  $D_{\text{max}}$  is the maximum number of documents supported by the scheme), Theorem 4.1 tells us that the complexity of updates is  $\Omega\left(\frac{1}{\log \log K}\right)$ . Finally, we explained that the actual lower bound for update complexity of forward private schemes was more likely to be  $\Omega\left(\frac{\log K}{\log |\sigma|}\right)$ , which tightly translates to  $\Sigma\phi\phi\phi$  when  $|\sigma| = K$ .

This justifies the optimality claim on the update complexity made higher, at the beginning of the section, and in  $\Sigma\phi\phi\phi$ ' name itself.

#### 4.5.7 Outsourcing the Client's State

One major downside of  $\Sigma\phi\phi\phi$  is the large storage needed on the client side: we need  $\mathcal{O}(K \log D_{\text{max}})$  bits to store the counters. It would be nice to reduce this storage, and even to achieve constant client

storage. And this directly raises the question of the existence of a forward private scheme with constant client storage and optimal updates ( $\mathcal{O}(\log K)$  update complexity).

**Storing the state in an ORAM.** It is very important that the outsourcing of  $\Sigma\text{of}\phi\text{os}'$  and Diana's counter map does not leak that the same counter was accessed when a search query on  $w$  is followed by an update query on  $w$ . A natural way to avoid that is to put the counter map in an ORAM, as every access would be hidden.

Let us consider using Path ORAM [SDS+13a]. We want to outsource  $\Theta(K)$  blocks, and with Path ORAM, this will imply a  $\Theta(\log^2 K)$  computational overhead, with a  $\mathcal{O}(\log K)$  client state which can itself be downloaded (and later re-uploaded) to the server for every counter map access (cf. [SDS+13b, Section 6.2]).

If we use the construction of Kushilevitz *et al.* [KLO12] instead, we directly end up with a construction with constant client state and  $\mathcal{O}\left(\frac{\log^2 K}{\log \log K}\right)$  update complexity.

Note that, using this construction, we are unable to tightly match the lower bound of Theorem 4.1 (it is unclear if the  $\mathcal{O}(\log K)$  overhead offered by Path ORAM when the blocks are large enough really applies here as we cannot generically evaluate the size of the blocks in our case), but that any improvement in the ORAM constructions would lead to an improvement in our case. Actually, if an ORAM scheme is shown to tightly meet the lower bound of [GO96], we will be able to construct a tight forward private SSE scheme by using  $\Sigma\text{of}\phi\text{os}$  and this ORAM to outsource the client's state.

**Batching ORAM updates.** A practical issue we might encounter by using ORAM to store the counter map is that we pay for the update obliviousness even during searches, which implies having a non-optimal search algorithm (e.g.  $\mathcal{O}(n_w + \log^2 K)$  complexity instead of  $\mathcal{O}(n_w)$ ). To reduce this cost, we can batch the ORAM updates induced by the SSE search queries, as done by Miers and Mohassel in [MM17].

More precisely, when reading the counter map during a search query, the client will keep the ORAM leaf in a list of *deferred reads*, and, during the next Update query, the client will first execute these deferred reads on the ORAM, before reading and modifying the counter associated with the updated keyword. Doing this will leak that some search queries are repeated, but that will be the case anyway given  $\Sigma\text{of}\phi\text{os}'$  leakage profile.

## 4.6 Diana: Forward-Secure SSE with Very Low Overhead

In this section, we describe a generic way to construct forward-private searchable encryption from constrained PRFs on  $\mathbb{N}$  with respect to the range family of circuits  $\mathcal{C} = \{C_c | C_c(x) = 1 \Leftrightarrow 0 \leq x \leq c\}$  (cf. Section 2.3.2 for the full description of constrained PRFs). We will see that  $\Sigma\text{of}\phi\text{os}$  can be seen as an instantiation of this scheme, and then provide a much more efficient one based on the GGM PRF [GGM84], which we call Diana.

### 4.6.1 FS-RCPRF: Forward-Secure SSE from Range Constrained PRFs

Let  $\tilde{F} : \{0, 1\}^\lambda \times \{0, \dots, n_{\max}\} \rightarrow \{0, 1\}^\mu$  be a constrained PRF with respect to the class of range circuits  $\mathcal{C}$  defined above. Also, let  $F$  be a  $2\lambda$ -bit PRF, and  $H_1$  and  $H_2$  two hash functions modeled as two random oracles with  $H_1 : \{0, 1\}^\lambda \times \{0, 1\}^\mu \rightarrow \{0, 1\}^\lambda$ , and  $H_2 : \{0, 1\}^\lambda \times \{0, 1\}^\mu \rightarrow \{0, 1\}^\ell$ . Algorithm 4.6 describes FS-RCPRF, a forward-secure scheme based on the range-constrained PRF  $\tilde{F}$ . The simple idea behind FS-RCPRF is that update tokens for entries matching keyword  $w$  are generated

using  $\tilde{F}$  in counter mode, where the counter is incremented every time a new entry matching  $w$  is inserted. Then, during search, the client gives to the server the constrained key allowing only for the evaluation of  $\tilde{F}$  on  $\{0, \dots, n_w\}$ . The resulting scheme can be seen as a generalization of the dynamic scheme of Cash *et al.* [CJJ+14], where, during the search, the client gives to the server the constrained key of  $w$  instead of the master key  $K_w$ .

---

**Algorithm 4.6** FS-RCPRF: Forward private SSE scheme from range-constrained PRF  $\tilde{F}$ .

---

Setup()

- 1:  $K_\Sigma \xleftarrow{\$} \{0, 1\}^\lambda$ ,  $\mathbf{W}$ , EDB  $\leftarrow$  empty map
- 2: **return** (EDB,  $K_\Sigma$ ,  $\mathbf{W}$ )

Search( $K_\Sigma$ ,  $w$ ,  $\sigma$ ; EDB)

*Client:*

- 1:  $K_w || K'_w \leftarrow F_{K_\Sigma}(w)$
- 2:  $c \leftarrow \mathbf{W}[w] \quad \triangleright c = n_w - 1$
- 3: **if**  $c = \perp$
- 4:   **return**  $\emptyset$
- 5:  $ST \leftarrow \tilde{F}.\text{Constrain}(K_w, C_c) \quad \triangleright C_c$  is the circuit evaluating to 1 on  $\{0, \dots, c\}$
- 6: Send  $(K'_w, ST, c)$  to the server.

*Server:*

- 7: **for**  $i = c$  **to** 0 **do**
- 8:    $T_i \leftarrow \tilde{F}(ST, i)$
- 9:    $UT_i \leftarrow H_1(K'_w, T_i)$
- 10:    $e \leftarrow \text{EDB}[UT_i]$
- 11:    $\text{ind} \leftarrow e \oplus H_2(K'_w, T_i)$
- 12:   Output each ind
- 13: **end for**

Update( $K_\Sigma$ , add,  $w$ , ind,  $\sigma$ ; EDB)

*Client:*

- 1:  $K_w || K'_w \leftarrow F(K_\Sigma, w)$
- 2:  $c \leftarrow \mathbf{W}[w]$
- 3: **if**  $c = \perp$
- 4:    $c \leftarrow -1$
- 5:  $T_w^{c+1} \leftarrow \tilde{F}(K_w, c+1)$
- 6:  $\mathbf{W}[w] \leftarrow c+1$
- 7:  $UT_{c+1} \leftarrow H_1(K'_w, T_w^{c+1})$
- 8:  $e \leftarrow \text{ind} \oplus H_2(K'_w, T_w^{c+1})$
- 9: Send  $(UT_{c+1}, e)$  to the server.

*Server:*

- 10: EDB[ $UT_{c+1}$ ]  $\leftarrow e$

The intuition for the security of FS-RCPRF is simple: as the adversary only gets to see the CPRF keys for ranges corresponding to already inserted entries during searches, he cannot predict the evaluation of the PRF for inputs outside of these ranges, and in particular for newly inserted entries. Hence updates leak no information. Theorem 4.4 states the formal security of FS-RCPRF.

**Theorem 4.4** (Adaptive security of FS-RCPRF). *We recall that  $L_{\text{FP}} = (L_{\text{FP}}^{\text{Srch}}, L_{\text{FP}}^{\text{Updt}})$  is defined as:*

$$L_{\text{FP}}^{\text{Srch}}(w) = (\text{sp}(w), \text{UpHist}(w))$$

$$L_{\text{FP}}^{\text{Updt}}(\text{add}, w, \text{ind}) = \perp.$$

FS-RCPRF is  $L_{\text{FP}}$ -adaptively-secure. More precisely, for any polynomial-time adversary  $A$  encrypting a database of a most  $N$  entries, with a most  $K$  distinct keywords, and  $q_{\text{RO}}$  queries to the random oracles  $H_1$  and  $H_2$ , there exists polynomial-time adversaries  $B_1$  and  $B_2$ , and a simulator  $S$  such that

$$\text{Adv}_{\text{FS-RCPRF}, S, \mathcal{L}, A}^{\text{SSE-sim}}(\lambda) \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda) + 2N \cdot \text{Adv}_{\tilde{F}, B_2}^{\text{cprf}}(\lambda) + \frac{2Nq_{\text{RO}}}{2^\mu}.$$

*Proof.* The proof proceeds using a hybrid argument, by game hopping, starting from the real-world game  $\text{SSEReal}_A^{\text{FS-RCPRF}}(\lambda)$ .

**Game  $G_0$ .** This game is exactly the real world SSE security game  $\text{SSEReal}$ .

$$\mathbb{P}[\text{SSEReal}_{\text{FS-RCPRF}}^A(\lambda) = 1] = \mathbb{P}[G_0(1^\lambda) = 1]$$

**Game  $G_1$ .** In this game, we replace the calls to the PRF  $F$  by picking a new random output every time a previously unseen keyword is used. These strings are stored in a table to be reused every time  $F$  is again queried on  $w$ . The adversarial distinguishing advantage between  $G_0$  and  $G_1$  is exactly the distinguishing advantage for the PRF  $F$ : we can build a reduction  $B_1$  making at most  $W$  calls on  $F$  such that

$$\mathbb{P}[G_0(1^\lambda) = 1] - \mathbb{P}[G_1(1^\lambda) = 1] \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda).$$

**Game  $G_2$ .** In  $G_2$ , the update tokens  $UT$  are generated as random strings, instead of using  $H_1$ . These strings will then be programmed in the random oracle to ensure that  $H_1(K_w, T_c(w)) = UT_c(w)$ .

Algorithm 4.7 formally describes  $G_2$ , together with the intermediate game  $\widetilde{G}_2$ , which includes the additional boxed lines. The calls to the random oracle  $H_1$  are made explicit, and the game keeps track of these using the table  $H_1$ . It allows us to program the RO during the Search algorithm (cf. line 6). Note that, for the table Key, if an entry is accessed for the first time, a new random value is picked and placed in the table.

Also,  $G_2$  and  $\widetilde{G}_2$  make some bookkeeping of the tokens  $T_c$ . This bookkeeping allows for exactly programming  $H_1$  when it is queried by the adversary on a valid  $(K'_w, T_w^c)$  couple, at line 5.

Hence,  $H_1$ 's behavior in  $\widetilde{G}_2$  and  $G_1$  are perfectly indistinguishable, and:

$$\mathbb{P}[\widetilde{G}_2(1^\lambda) = 1] = \mathbb{P}[G_1(1^\lambda) = 1].$$

To find the distinguishing advantage between  $\widetilde{G}_2$  and  $G_2$ , we use the *identical-until-bad* approach:  $\widetilde{G}_2$  and  $G_2$  are identical until the flag bad is set to true:

$$\mathbb{P}[\widetilde{G}_2(1^\lambda) = 1] - \mathbb{P}[G_2(1^\lambda) = 1] \leq \mathbb{P}[\text{bad is set to true in } \widetilde{G}_2].$$

We are going to show that when the adversary is able to set bad at true, he will break the CPRF security game, by constructing a reduction  $B_2$  from a distinguisher  $A$  inserting  $N$  keyword/document pairs in the database.  $B_2$  first guesses the pair  $(w^*, c^*)$  for which bad will be set to true for the first time, by querying  $H_1$  on  $(K'_{w^*}, T_{c^*})$  (i.e. by pre-computing  $\text{UT}[w^*, c^*]$ ), among the  $N$  possible pairs. For all keyword  $w \in W \setminus \{w^*\}$ ,  $B_2$  behaves exactly as game  $\widetilde{G}_2$ . Note that if  $w^*$  has been correctly guessed, then it means that  $B_2$  behaves exactly as game  $G_2$  for these keywords. For  $w^*$ ,  $B_2$  will call its CPRF game oracles (cf. Section 2.3.2) to generate the tokens as follows:

$$\begin{aligned} T_i(w^*) &\leftarrow \text{Eval}(i) \text{ for } 0 \leq i < c^*, \\ T_i(w^*) &\leftarrow \text{Challenge}(i) \text{ for } i \geq c^*, \\ ST(w^*) &\leftarrow \text{Constrain}(C_{n_{w^*}}). \end{aligned}$$

By closely looking at  $G_2$ 's code, we see that bad is set to true only if  $H_1$  is queried on  $(K'_{w^*}, T_c)$  for a value  $c$  that has never been queried to  $\text{Eval}$ , and for which there is no  $C_{c'}$  with  $c' \geq c$  on which  $\text{Constrain}$  has been queried. It implies that all the queries to  $\text{Challenge}$  are valid, and that the value  $T_{c^*}$  raising bad to true is indistinguishable from random by definition of CPRF security. Also, if  $A$  makes  $q_{\text{RO}}$  queries to the random oracle (apart from the ones already needed by the execution of the



---

**Algorithm 4.7** Games  $G_2$  and  $\widetilde{G}_2$  Boxed code is included in  $\widetilde{G}_2$  only.

---

Setup()	$Update(K_\Sigma, \text{add}, w, \text{ind}, \sigma; \text{EDB})$
1: $\text{bad} \leftarrow \text{false}$ 2: $\text{Key}, \mathbf{W}, \text{EDB} \leftarrow \text{empty map}$ 3: <b>return</b> $(\text{EDB}, (\text{Key}, \mathbf{W}))$	<i>Client:</i> 1: $K_w    K'_w \leftarrow \text{Key}[w]$ 2: $(T_0, \dots, T_c, c) \leftarrow \mathbf{W}[w]$ 3: <b>if</b> $c = \perp$ <b>then</b> $c \leftarrow -1$ 4: $T_w^{c+1} \leftarrow \widetilde{F}(K_w, c+1)$ 5: $\mathbf{W}[w] \leftarrow (T_0, \dots, T_c, T_{c+1}, c+1)$ 6: $UT_{c+1} \leftarrow \{0, 1\}^\lambda$ 7: <b>if</b> $H_1(K'_w, T_{c+1}) \neq \perp$ <b>then</b> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin: 2px;">             bad <math>\leftarrow</math> true, <math>UT_{c+1} \leftarrow H_1(K'_w, T_{c+1})</math> </div> 9: <b>end if</b> 10: $UT[w, c+1] \leftarrow UT_{c+1}$ 11: $e \leftarrow \text{ind} \oplus H_2(K'_w, T_w^c)$ 12: <b>Send</b> $(UT_{c+1}, e)$ to the server.
$Search(K_\Sigma, w, \sigma; \text{EDB})$ <i>Client:</i> 1: $K_w    K'_w \leftarrow \text{Key}[w]$ 2: $(T_0, \dots, T_c, c) \leftarrow \mathbf{W}[w]$ 3: <b>if</b> $(T_0, \dots, T_c, c) = \perp$ <b>then return</b> $\emptyset$ 4: $[(u_0, \text{ind}_0), \dots, (u_c, \text{ind}_c)] \leftarrow \text{UpHist}(w)$ <div style="text-align: right; margin-right: 20px;"><math>\triangleright</math> In the order of updates</div> 5: <b>for</b> $i = 0$ <b>to</b> $c$ <b>do</b> 6: $H_1(K'_w, T_i) \leftarrow UT[w, i]$ 7: <b>end for</b> 8: $ST \leftarrow \widetilde{F}.\text{Constrain}(K_w, C_c)$ 9: <b>Send</b> $(K'_w, ST, c)$ to the server. <i>Server:</i> 10: <b>for</b> $i = c$ <b>to</b> $0$ <b>do</b> 11: $T_i \leftarrow \widetilde{F}(ST, i)$ 12: $UT_i \leftarrow H_1(K'_w, T_i)$ 13: $e \leftarrow \text{EDB}[UT_i]$ 14: $\text{ind} \leftarrow e \oplus H_2(K'_w, ST_i)$ 15: <b>Output</b> each ind 16: <b>end for</b>	<i>Server:</i> 13: $\text{EDB}[UT_{c+1}] \leftarrow e$ $H_1(k, t)$ 1: $v \leftarrow H_1(k, t)$ 2: <b>if</b> $v = \perp$ <b>then</b> 3: $v \xleftarrow{\$} \{0, 1\}^\lambda$ 4: <b>if</b> $\exists w, c$ s.t. $k = \text{Key}[w]$ <b>and</b> $t = T_c \in \mathbf{W}[w]$ <b>then</b> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin: 2px;">             bad <math>\leftarrow</math> true, <math>v \leftarrow UT[w, c]</math> </div> 6: <b>end if</b> 7: $H_1(k, st) \leftarrow v$ 8: <b>end if</b> 9: <b>return</b> $v$

---

game), as  $T_{c^*}$  is uniformly random, the probability that  $H_1$  was called on  $(K'_{w^*}, T_{c^*})$  is  $q_{\text{RO}} \cdot 2^{-\mu}$ . Hence,

$$\mathbb{P}[\text{bad is set to true in by querying } (K'_{w^*}, T_{c^*})] \leq \text{Adv}_{\widetilde{F}, B_2}^{\text{cprf}}(\lambda) + \frac{q_{\text{RO}}}{2^\mu},$$

and, as guessing the pair  $(w^*, c^*)$  implies a  $N$  loss factor in the advantage of the reduction from distinguishing  $G_2$  and  $\widetilde{G}_2$  to the game of setting bad to true,

$$\begin{aligned} \mathbb{P}[G_1(1^\lambda) = 1] - \mathbb{P}[G_2(1^\lambda) = 1] &= \mathbb{P}[\widetilde{G}_2(1^\lambda) = 1] - \mathbb{P}[G_2(1^\lambda) = 1] \\ &\leq N \cdot \text{Adv}_{\widetilde{F}, B_2}^{\text{cprf}}(\lambda) + \frac{Nq_{\text{RO}}}{2^\mu}. \end{aligned}$$

**Game  $G_3$ .** In game  $G_3$ , we do exactly as in  $G_2$ , but for  $H_2$ :

$$\mathbb{P}[G_2(1^\lambda) = 1] - \mathbb{P}[G_3(1^\lambda) = 1] \leq N \cdot \text{Adv}_{\widetilde{F}, B_2}^{\text{cprf}}(\lambda) + \frac{Nq_{\text{RO}}}{2^\mu}.$$

**Algorithm 4.8** Game  $G_4$ .Setup()

- 1:  $u \leftarrow 0$
- 2:  $\mathbf{W}, \text{EDB} \leftarrow \text{empty map}$
- 3: **return**  $(\text{EDB}, \emptyset, \mathbf{W})$

Search( $K_\Sigma, w, \sigma; \text{EDB}$ )

- Client:*
- 1:  $K_w || K'_w \leftarrow \text{Key}[w]$
  - 2:  $c \leftarrow \mathbf{W}[w]$
  - 3:  $[(u_0, \text{ind}_0), \dots, (u_c, \text{ind}_c)] \leftarrow \text{UpHist}(w)$
  - 4: **if**  $c = \perp$  **then return**  $\emptyset$
  - 5: **for**  $i = 0$  **to**  $c$  **do**
  - 6:   Program  $H_1$  s.t.  $H_1(K_w, \tilde{F}(K_w, i)) \leftarrow \text{UT}[u_i]$
  - 7:   Program  $H_2$  s.t.  $H_2(K_w, \tilde{F}(K_w, i)) \leftarrow \mathbf{e}[u_i] \oplus \text{ind}_i$
  - 8: **end for**
  - 9:  $ST \leftarrow \tilde{F}.\text{Constrain}(K_w, C_c)$
  - 10: Send  $(K'_w, ST, c)$  to the server.

Update( $K_\Sigma, \text{add}, w, \text{ind}, \sigma; \text{EDB}$ )

- Client:*
- 1:  $\text{UT}[u] \xleftarrow{\$} \{0, 1\}^\mu$
  - 2:  $\mathbf{e}[u] \xleftarrow{\$} \{0, 1\}^\lambda$
  - 3: Send  $(\text{UT}[u], \mathbf{e}[u])$  to the server.
  - 4:  $u \leftarrow u + 1$

**Game  $G_4$ .** Game  $G_4$ , (cf. Algorithm 4.8) keeps track of the randomly generated string  $UT$  and  $e$  in dedicated tables: each time an update is performed, new randomness is appended to the tables and then returned to the server. Then, in Search, the random oracles are programmed as in  $G_3$ , so to have consistent results. To do so,  $G_4$  uses the information from  $\text{UpHist}(w)$  to know which update corresponds to which keyword-document pair.

We got rid of the server's part in the protocols as it is unchanged: these are single round-trip protocols and the removed lines do not influence the client's transcript. Finally, we have

$$\mathbb{P}[G_3(1^\lambda) = 1] - \mathbb{P}[G_4(1^\lambda) = 1] = 0.$$

**The simulator.** The simulator can directly be derived from  $G_4$ 's code. We just have to replace direct uses of the searched keyword  $w$  by  $\min \text{sp}(w)$ .  $G_4$  and  $\text{SSEIDEAL}_{S, \mathcal{L}_\Sigma}$  will then be identical games, the only difference being that, instead of the keyword  $w$ ,  $S$  uses the counter  $\bar{w} = \min \text{sp}(w)$  uniquely mapped from  $w$  using the leakage function.

$$\mathbb{P}[G_4(1^\lambda) = 1] - \mathbb{P}[\text{SSEIDEAL}_{\text{FS-RCPRF}, S, L_{\text{FP}}}^A(1^\lambda) = 1] = 0.$$

**Conclusion.** By combining all the contributions from all the games, there exists 2 adversaries  $B_1$  and  $B_2$  such that

$$\begin{aligned} \mathbb{P}[\text{SSEREAL}_{\text{FS-RCPRF}}^A(1^\lambda) = 1] - \mathbb{P}[\text{SSEIDEAL}_{\text{FS-RCPRF}, S, L_{\text{FP}}}^A(1^\lambda) = 1] \\ \leq \text{Adv}_{F, B_1}^{\text{prf}}(\lambda) + 2N \cdot \text{Adv}_{\tilde{F}, B_2}^{\text{cprf}}(\lambda) + \frac{2Nq_{\text{RO}}}{2^\mu}. \end{aligned}$$

□

The adaptive security of FS-RCPRF is shown in the random oracle model (ROM), but the ROM is not needed for non-adaptive security.

Finally, note that the client's state can be outsourced on the server, as for  $\Sigma\phi\phi\phi$  (cf. Section 4.5.7).

**Reinterpreting  $\Sigma\phi\phi\sigma$  with constrained PRFs.** The  $\Sigma\phi\phi\sigma$  construction is based on the iteration of a trapdoor permutation  $\pi$  to generate the update tokens in a way that prevents the server from predicting them.  $\Sigma\phi\phi\sigma$  can be reinterpreted using our framework by constructing a TDP-based range-constrained PRF  $\tilde{F}_\Sigma$ .

The master key  $\tilde{F}_\Sigma$  is composed of an RSA key SK and an element  $ST_0 \in \mathbb{Z}_N$  where each can be pseudo-randomly generated from a random  $\lambda$ -bit key.  $\tilde{F}((\text{SK}, ST_0), c) = H(\pi_{\text{SK}}^{-c}(ST_0))$  where  $\pi^{-c}$  is the  $c$ -fold iteration of  $\pi^{-1}$ , and  $H$  is a hash function modeled as a random oracle. The constrain algorithm will then be the following (we identify the circuit constraining to the range  $\{0, \dots, n\}$  with the integer  $n$ ):

$$\tilde{F}.\text{Constrain}((\text{SK}, ST_0), n) = (\text{PK}, \pi_{\text{SK}}^{-n}(ST_0), n) = (\text{PK}, ST_n, n).$$

Finally, the constrained evaluation function is

$$\tilde{F}.\text{Eval}((\text{PK}, ST_n, n), c) = H(\pi_{\text{PK}}^{n-c}(ST_c)).$$

We could easily reduce the constrained-PRF security of  $\tilde{F}$  to the hardness of the RSA assumption, and directly deduce the security of  $\Sigma\phi\phi\sigma$  from Theorem 4.4.

Yet, there is a subtle difference between  $\Sigma\phi\phi\sigma$  and this RSA-based instantiation of FS-RCPRF:  $\Sigma\phi\phi\sigma$  uses a single key pair for the entire scheme, while here, we would have one key pair per keyword. Seen otherwise,  $\Sigma\phi\phi\sigma$  uses a single constrained PRF, which is constrained both on ranges and on keywords. We could have written such a generalization of FS-RCPRF, but it would have made the proof more complicated than the one of Theorem 4.4, without clear conceptual benefits.

#### 4.6.2 Diana, a GGM Instantiation of FS-RCPRF

In this section we present a range-constrained PRF and then use it to instantiate FS-RCPRF.

We can easily construct a simple and efficient range-constrained PRF from the tree-based GGM PRF [GGM84]. This instantiation has been described by Kiayias *et al.* [KPTZ13] and is called best range cover (BRC).

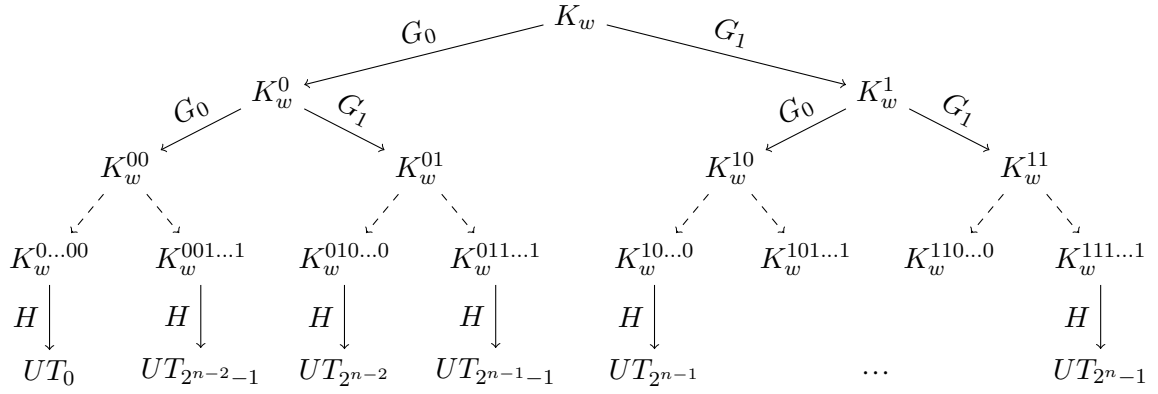
Let  $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$  be a pseudo-random generator (PRG),  $G_0(k)$  and  $G_1(k)$  be respectively the first and second half of  $G(k)$ . The GGM PRF on  $n$ -bit integers is defined as

$$F_K(x) = G_{x_0}(G_{x_1}(\dots(G_{x_{n-1}}(K))))$$

where  $x_{n-1} \dots x_0$  is the binary representation of  $x = \sum_{i=0}^{n-1} 2^i x_i$ . The leaves of the tree are the output values of  $F$ , and they can be labeled according to the corresponding input, and the partial evaluation of  $F$  (*i.e.* the iterated evaluation of  $G$ , but only on the first  $m < n$  bits) are the inner nodes of the tree.

To constrain  $F$  to the input range  $[0, c-1]$ , we generate the nodes of the tree covering exactly the leaves with labels in  $[0, c-1]$ . In practice if the binary representation of  $c$  is  $c_{n-1} \dots c_0$ , the punctured key would be  $\{G_0(G_{c_{i-1}}(\dots(G_{c_{n-1}}(K))))\}$  for all  $i$  such that  $c_i = 1$ . Figure 4.2 represents the token generation in Diana.

We use the above range CPRF to instantiate FS-RCPRF and call this instantiation Diana. Note that, Diana is almost identical to the ARX-EQ scheme [BBP16]. However ARX-EQ was not formally proven, and FS-RCPRF provides a more general framework on how to construct forward-private SSE schemes.



**Figure 4.2** – Token generation in Diana for a keyword  $w$  and a tree of depth  $n$ . Each operation (represented by arrows) is one way.

Let us analyze the efficiency of Diana. Updates require  $\mathcal{O}(\log n_{\max})$  computation from the client, where  $n_{\max}$  is the maximum number of entries matching a keyword: the CPRF computes a tree’s leaf from its root. Similarly, during search, the server has to compute all the leaves of the tree within a given range. This can be done efficiently in  $\mathcal{O}(n_w)$  calls to the PRG, where  $n_w$  is the number of matches on search keyword  $w$ : there are  $\mathcal{O}(n_w)$  tree nodes to compute in total and each node can be generated using a single PRG call. In terms of communication complexity, Diana is optimal for updates, and sends  $\mathcal{O}(\log n_w)$  tree nodes during a search query.

In theory, this is worse than  $\Sigma\phi\phi\phi\phi$ ’s optimal computational and communication complexity, but, as we will see in Section 4.7, Diana uses symmetric primitives that are much faster than  $\Sigma\phi\phi\phi\phi$ ’s RSA. Also, since nodes in the tree will be 128 bit keys, we can set  $n_{\max}$  to  $2^{32}$  and still have search tokens only twice as big as  $\Sigma\phi\phi\phi\phi$ ’s 2048 bits tokens.

## 4.7 Implementation and Evaluation of $\Sigma\phi\phi\phi\phi$ and Diana

In this section, we describe an implementation of  $\Sigma\phi\phi\phi\phi$  and Diana, report and analyze the performance of these schemes.

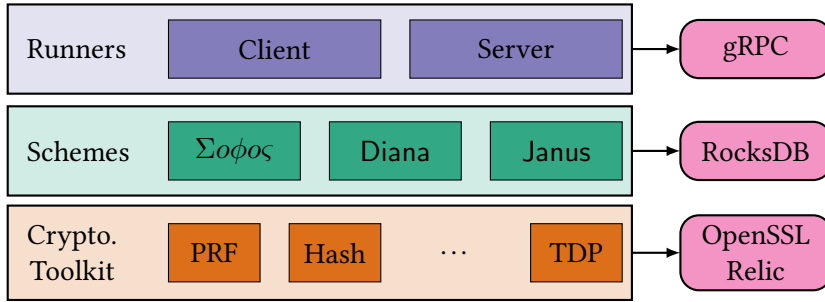
### 4.7.1 Implementation Details

Both  $\Sigma\phi\phi\phi\phi$  and Diana have been implemented in C/C++ (and a little bit of assembly), representing about 5000 lines of code for the schemes *per se*, and 14 000 lines for the cryptographic toolkit (of which not all components are used in the implementation of  $\Sigma\phi\phi\phi\phi$  and Diana). The code is open source and freely accessible [Bos17]

The primitives we used are the following. For the trapdoor permutation, we used RSA. The RSA implementation uses OpenSSL’s BigNum library. The PRF  $F$  and the keyed hash function  $H$  are instantiated using HMAC, and we chose Blake2b [SA15] as the underlying hash function. For the GGM range-constrained PRF  $\tilde{F}$ , we used AES in counter mode for the pseudo-random generator  $G$ . The keyed hash function  $H$  used in Diana (and Diana only) is the AES block cipher [Pub01] used in Matyas-Meyer-Oseas mode [PGV94].

Aside from cryptographic components, we needed a persistent key-value store, and an RPC

framework. We chose RocksDB [Fac17] for the former, and gRPC [Goo17] for the latter, for their ease of use and their performance. Figure 4.3 presents the architecture of our implementation.



**Figure 4.3** – Software architecture of our implementation of symmetric searchable encryption schemes (OpenSSE). This diagram also includes the scheme Janus, which will be presented in Chapter 5.

**Parameters.** Cryptographic keys are 128 bits long for symmetric primitives, and we chose to use 2048 bits RSA keys, with a public exponent fixed to 3.

Our system can easily be scaled to support databases with up to  $2^{64}$  keyword/document pairs, but to avoid unnecessary overhead for our experiments using much smaller database, we fixed  $N \leq 2^{48}$ ,  $K \leq 2^{48}$ , and the maximum matching documents per keyword  $n_{\max} \leq 2^{32}$ . We set  $\mu$ , the length of the update tokens, to 128 bits, leading to an incorrectness probability of  $2^{-32}$  for the maximum size database.

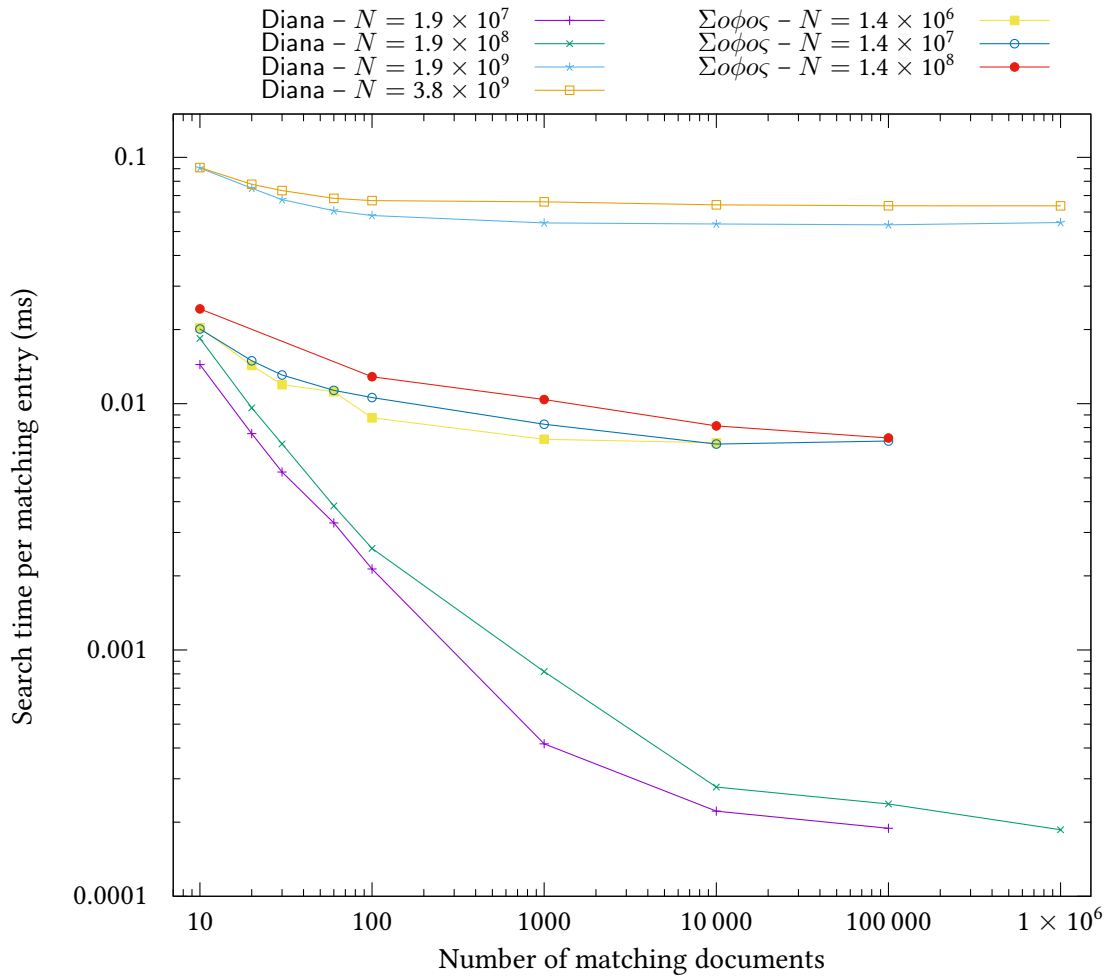
### 4.7.2 Experimental Setting

All the experiments were run on a desktop computer with a single Intel Core i7 4790K 4.00 GHz CPU (with 8 logical cores), 16 GB of RAM, a 250 GB Samsung 850 EVO SSD, running on OS X.10.

We tested our schemes on randomly generated databases of different sizes. The characteristics of the databases are given in Table 4.2, as well as the size of the encrypted database, and the size of the client’s state (*i.e.* the size of  $\mathbf{W}$ ).

**Table 4.2** – Size of the evaluation databases.

(a) $\Sigma\phi\phi\phi\phi$ ’ evaluation databases. The last database is the English Wikipedia.				(b) Diana’s evaluation databases.			
$K$	$N$	$\mathbf{W}$	EDB	$K$	$N$	$\mathbf{W}$	EDB
$2.33 \times 10^4$	$1.4 \times 10^6$	572 kB	64.0 MB	$2.22 \times 10^5$	$1.9 \times 10^7$	4.6 MB	615 MB
$2.13 \times 10^5$	$1.4 \times 10^7$	4.81 MB	512 MB	$2.68 \times 10^6$	$1.9 \times 10^8$	46 MB	6.3 GB
$2.11 \times 10^6$	$1.4 \times 10^8$	64.2 MB	5.25 GB	$2.18 \times 10^7$	$1.9 \times 10^9$	365 MB	47 GB
$4.60 \times 10^6$	$1.39 \times 10^8$	128 MB	5.25 GB	$4.29 \times 10^7$	$3.8 \times 10^9$	720 MB	95 GB



**Figure 4.4** – Performance of  $\Sigma\omicron\phi\omicron\varsigma$ ' and Diana's and search protocol (server side). log-log scale.

### 4.7.3 Results and Interpretation

The performance results of keyword searches are presented in Figure 4.4. This figure only accounts for the running time on the server side, without the RPC costs – we focus on the core of the algorithm. The timings given here are means of the elapsed search time per found document, taken over between 1000 queries (for queries matching a small number of results relatively to the size of the dataset) to 30 queries. This allows us to exhibit some real-world side effects in our implementation.

The first thing we can notice in the timings, is that the larger the result set, the faster the search algorithm (again, on a per matching document basis). We explain that by the cost of multi-threading, and by the storage latency: even if the RSA or GGM pseudo-random function computations are fully parallelized, adding an index to the result list is not, and accessing the disk induces some wait. Hence, at the beginning of the search algorithm, RSA operations will not be fully interleaved with disk accesses (like they are for a sufficiently large result set), we will pay for the latency induced by mutexes and storage accesses. Also, in the case of small result sets, some one-time costly operations (such as creating threads) are not amortized, and in some cases, using several threads might actually hinder the performance.

**Cryptography vs. locality vs. OS caching.** Figure 4.4 displays three groups of graphs. At the bottom, we have Diana with the smallest databases, then  $\Sigma\phi\phi\phi$  with databases smaller or equivalent to the smallest datasets of Diana, and at the top, Diana with the two largest datasets. Also, the performances of  $\Sigma\phi\phi\phi$  and Diana on small result sets and small databases is very similar, but diverge once the result set grows. How can we explain that?

The latter (similar performance for small result set followed by a divergence of the performance) can easily be explained similarly as above, by the fact that some one-time costly operations get amortized when the number of matches grows.

Also, the discrepancy between the efficiency of  $\Sigma\phi\phi\phi$  and Diana search algorithm for databases of similar size, and result sets of identical size can easily be explained by the performance gap between the RSA’s public key operations and hardware-accelerated AES.

The last interrogation is about the (seemingly) bad performance of Diana with large database compared to  $\Sigma\phi\phi\phi$  and Diana with smaller datasets. This cannot be explained by the algorithms’ difference, as, from that point of view, Diana should perform a lot better than  $\Sigma\phi\phi\phi$  in practice – this was confirmed by the experiments, as we just mentioned. Also, this would not explain the fact that Diana on large datasets is slower by two to three orders of magnitude than Diana on small datasets.

One important thing to notice is that all the experimental encrypted databases, except the two largest ones for Diana, hold in the RAM of the testing computer. Indeed, for the former ones, the operating system cached all the encrypted database – as confirmed by using the `vmtouch` tool [Dou17] – and many non-volatile-storage-induced performance issues disappeared, among the first being the lack of locality in the accesses.

With in-memory storage, the need of local accesses is no longer necessary, as the bottleneck becomes indeed the cryptographic operations. When the on-disk storage actually has to be accessed, this is not true anymore: the latency of the storage is a lot higher than the cryptographic operations overhead, even for asymmetric cryptography.

Here, we have a blatant example of the influence of storage locality on the performance of SSE schemes. Hence, for large datasets IO costs outweigh the cost of cryptographic primitives, making the latter “almost free”. In particular, for larger datasets than the ones tested here,  $\Sigma\phi\phi\phi$  would encounter similar IO bottleneck, and would perform (almost) as well as Diana on large inputs.

Finally, the rise of high-performance low-latency storage with SSDs gives a hope that non-locality would be less of a burden in the future. Namely, the data sheet of the SSD used in our experiments announces 98 000 random 4 kB reads per second [Sam16], which is of the order of what we experienced (about 16 000 fetched entries per second), while some high-end SSDs are suppose to achieve up to 430 000 random reads per second, with a 20  $\mu$ s latency [Int].

**Update throughput.** The major practical drawback of  $\Sigma\phi\phi\phi$  is its update throughput. Indeed, it can only insert 4300 new entries per second on our test machine. The bottleneck is clearly the computation of the new search token used to derive the update token. In particular, we pay for the fact that private key operations are a lot more expensive than public key operations (to a factor 30 on our computer).

Diana completely solves this problem, achieving an update throughput of more than 174 000 entries per second. The bottleneck is no longer the cryptography, but the use of locks to ensure the thread-safety, and the networking costs.

## References

- [ACN+17] Gilad Asharov, T-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. *Oblivious Computation with Data Locality*. Cryptology ePrint Archive, Report 2017/772. <http://eprint.iacr.org/2017/772>. 2017 (cit. on pp. 69, 178).
- [ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. *Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations*. In: *48th ACM STOC*. Ed. by Daniel Wichs and Yishay Mansour. ACM Press, June 2016, pp. 1101–1114 (cit. on pp. 58, 59, 69, 71).
- [BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. *Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives*. In: *ACM CCS 17*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1465–1482 (cit. on pp. viii, ix, 10, 13, 65, 97).
- [Bos16] Raphael Bost. *Σοφος: Forward Secure Searchable Encryption*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1143–1154 (cit. on pp. viii, 9, 10, 12, 56, 65).
- [Bos17] Raphael Bost. *A State of the Art Single-Keyword Searchable Encryption Library in C/C++*. 2017. [Link](#). (Cit. on p. 88).
- [CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions*. In: *ACM CCS 06*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, Oct. 2006, pp. 79–88 (cit. on pp. 9, 42–45, 56, 72, 152).
- [CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. *Leakage-Abuse Attacks Against Searchable Encryption*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 668–679 (cit. on pp. 60, 66, 151–153, 156, 161, 162, 170–172).
- [CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. 10, 49, 56, 73, 83, 99, 129, 163, 170, 171).
- [CT14] David Cash and Stefano Tessaro. *The Locality of Searchable Symmetric Encryption*. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 351–368 (cit. on pp. 57, 58, 68).
- [Dou17] Doug Hoyte. *vmtouch - the Virtual Memory Toucher*. 2017. [Link](#). (Cit. on p. 91).
- [DP17] Ioannis Demertzis and Charalampos Papamanthou. *Fast Searchable Encryption With Tunable Locality*. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM. 2017, pp. 1053–1067 (cit. on pp. 58, 59, 68, 69).
- [Fac17] Facebook, Inc. *RocksDB: A Persistent Key-Value Store for Fast Storage Environments*. Sept. 2017. [Link](#). (Cit. on p. 89).
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. *How to Construct Random Functions (Extended Abstract)*. In: *25th FOCS*. IEEE Computer Society Press, Oct. 1984, pp. 464–479 (cit. on pp. 82, 87).



- [GM11] Michael T. Goodrich and Michael Mitzenmacher. *Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation*. In: *ICALP 2011, Part II*. Ed. by Luca Aceto, Monika Henzinger, and Jiri Sgall. Vol. 6756. LNCS. Springer, Heidelberg, July 2011, pp. 576–587 (cit. on pp. 59, 71, 130, 133, 147).
- [GMOT12] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. *Privacy-preserving group data access via stateless oblivious RAM simulation*. In: *23rd SODA*. Ed. by Yuval Rabani. ACM-SIAM, Jan. 2012, pp. 157–167 (cit. on pp. 59, 71).
- [GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. *TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption*. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563–592 (cit. on pp. 7, 8, 45, 69, 73, 99, 101, 161, 163, 170).
- [GO96] Oded Goldreich and Rafail Ostrovsky. “Software protection and simulation on oblivious RAMs”. In: *Journal of the ACM* 43.3 (1996), pp. 431–473 (cit. on pp. 7, 8, 51, 52, 56, 82).
- [Goo17] Google, Inc. *gRPC: A high performance, open-source universal RPC framework*. Sept. 2017. [Link](#). (Cit. on p. 89).
- [Int] Intel. *Intel®SSD 750 Series: Performance Unleashed*. [Link](#). (Cit. on p. 91).
- [KLO12] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. *On the (in)security of hash-based oblivious RAM and a new balancing scheme*. In: *23rd SODA*. Ed. by Yuval Rabani. ACM-SIAM, Jan. 2012, pp. 143–156 (cit. on p. 82).
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. *Delegatable pseudorandom functions and applications*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 669–684 (cit. on pp. 30, 87).
- [MM17] Ian Miers and Payman Mohassel. *IO-DSSE: Scaling Dynamic Searchable Encryption to Millions of Indexes By Improving Locality*. In: *NDSS 2017*. The Internet Society, 2017 (cit. on p. 82).
- [MMBC15] Tarik Moataz, Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. *Resizable Tree-Based Oblivious RAM*. In: *FC 2015*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. LNCS. Springer, Heidelberg, Jan. 2015, pp. 147–167 (cit. on p. 68).
- [PBP16] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. *Arx: A Strongly Encrypted Database System*. Cryptology ePrint Archive, Report 2016/591. <http://eprint.iacr.org/2016/591>. 2016 (cit. on pp. 9, 12, 87, 99).
- [PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. *Hash Functions Based on Block Ciphers: A Synthetic Approach*. In: *CRYPTO’93*. Ed. by Douglas R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 368–378 (cit. on p. 88).
- [Pub01] NIST FIPS Pub. “197: Advanced encryption standard (AES)”. In: *Federal information processing standards publication 197.441* (Nov. 2001), p. 0311. [Link](#). (Cit. on p. 88).
- [SA15] M-J. Saarinen and J-P. Aumasson. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. RFC 7693. RFC Editor, Nov. 2015 (cit. on p. 88).
- [Sam16] Samsung. *Samsung SSD 850 EVO Data Sheet, Rev.3.1*. May 2016. [Link](#). (Cit. on p. 91).

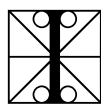
- [SDS+13a] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. *Path ORAM: an extremely simple oblivious RAM protocol*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 299–310 (cit. on pp. 7, 56, 82).
- [SDS+13b] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. *Path ORAM: An Extremely Simple Oblivious RAM Protocol*. Cryptology ePrint Archive, Report 2013/280. <http://eprint.iacr.org/2013/280>. 2013 (cit. on p. 82).
- [SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. viii–x, 10, 13, 45, 59, 67, 71, 73, 81, 97–99, 115, 119, 122, 130–133, 141, 147, 163, 170).
- [ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. *All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption*. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 2016, pp. 707–720 (cit. on pp. 10, 12, 60, 66, 76, 114, 152, 153, 156, 157).



I like the dreams of the future better than  
the history of the past

*Letters to John Adams* – THOMAS JEFFERSON

# Backward Privacy 5

 IN THE PREVIOUS CHAPTER, we studied the forward privacy property of searchable encryption schemes, which ensures that newly updated entries cannot be related to previous search results, until a new search query is performed. Another natural notion of privacy is that of *backward privacy*: search queries should not leak matching entries after they have been deleted. As such, backward privacy is closely related to secure deletion: we want to make sure that the server cannot recover deleted information.

In all the constructions studied earlier, deletions were supported using a sort of revocation list. In particular, the server still has access to the deleted information, while, paradoxically, the owner of the data, the client, does not. Achieving backward privacy is hence very important to withstand subpoena, offering a kind of deniable searchable encryption.

The backward privacy property had been informally introduced by Stefanov *et al.* in [SPS14], but was kept mainly overlooked until our work with Minaud and Ohrimenko [BMO17]. This chapter takes this work up by formulating formal security definitions for several forms of backward privacy, and building four dynamics schemes, Fides, Moneta, Diana<sub>del</sub> and Janus, satisfying these definitions with various efficiency tradeoffs, using forward private SSE schemes, and cryptographic primitives allowing for fine-grained control, such as puncturable PRF and puncturable encryption.

## Contents

---

<b>5.1</b>	<b>Definition of Backward Privacy</b>	<b>98</b>
<b>5.2</b>	<b>A Generic Two Round-trips Backward-Private Scheme</b>	<b>100</b>
5.2.1	Fides: A Baseline Forward and Backward Private SSE Scheme	102
5.2.2	Moneta: An (Almost) Strongly Backward Private Scheme	103
<b>5.3</b>	<b>Diana<sub>del</sub>: Backward Privacy from Range-Constrained and Puncturable PRFs</b>	<b>103</b>
<b>5.4</b>	<b>Janus: Weak Backward Privacy from Puncturable Encryption</b>	<b>104</b>
5.4.1	Puncturable Encryption	104
5.4.2	Incremental Puncture	106
5.4.3	The Janus Construction	107
5.4.4	Reducing the Storage Overhead	113
5.4.5	Security of Janus Against Weaker Adversaries	113
5.4.6	Performance of Janus	115

---

## 5.1 Definition of Backward Privacy

At a high level, forward privacy considered privacy of the database and earlier search queries during updates, while backward privacy captures privacy of the database and updates to it during search queries. In this section, we formally define these privacy properties.

Backward privacy limits the information on the updates affecting keyword  $w$  that the server can learn upon a search query on  $w$ . Informally, an SSE scheme is backward-private (or backward-secure) if, whenever a keyword/document pair  $(w, \text{ind})$  is added into the database and then deleted, subsequent Search queries on  $w$  do not reveal  $\text{ind}$  [SPS14]. Note that  $\text{ind}$  is revealed if a Search query is issued after  $(w, \text{ind})$  is added, and before it is deleted.

Hence, we could argue that backward-private schemes are those whose search leakage is only a (stateless) function of  $\text{DB}(w)$ , as this would only reveal information about documents currently in the database (and not the deleted ones). However, this is not enough, as, even though the search leakage is reduced to  $\text{DB}(w)$ , the update leakage could reveal the modified document/keyword pairs. A scheme with such leakage would reveal the indices of deleted documents, as the attacker could keep track of all the updated pairs, which is exactly what we want to prevent. As a consequence, in the security definitions, we must explicitly rule out such update leakage.

Moreover, obtaining a scheme with leakage that depends only on  $\text{DB}(w)$  would require hiding the pattern of updates as well as their number. Although hiding the former could be achieved, for example, using ORAM, this would result in expensive schemes. As a consequence, we define four flavors of backward privacy of decreasing strength, depending on how much metadata leaks about the inserted and deleted entries:

### 0. Strong backward privacy:

leaks only the documents *currently* matching  $w$ .

### I. Backward privacy with insertion pattern:

leaks the documents *currently* matching  $w$ , when they were inserted, and the total number of updates on  $w$ .

### II. Backward privacy with update pattern:

leaks the documents *currently* matching  $w$ , when they were inserted, and when all the updates on  $w$  happened (but not their content).

### III. Weak backward privacy:

leaks the documents *currently* matching  $w$ , when they were inserted, when all the updates on  $w$  happened, and which deletion update canceled which insertion update.

Let us demonstrate the differences between these notions with a simple example. Consider the following sequence of updates, in the order of arrival:  $(\text{add}, \text{ind}_1, \{w_1, w_2\})$ ,  $(\text{add}, \text{ind}_2, \{w_1\})$ ,  $(\text{del}, \text{ind}_1, \{w_1\})$ ,  $(\text{add}, \text{ind}_3, \{w_2\})$ . Let us consider the leakage for each definition after a search query on  $w_1$ . The first notion (notion 0) reveals only  $\text{ind}_2$ . The notion I reveals  $\text{ind}_2$  and that this entry was added at time 1. It also reveals that there were a total of 3 updates for  $w_1$ . The notion II, additionally reveals that updates on  $w_1$  happened at time 1, 2, and 3. Finally, the last definition also reveals that the index that was added for  $w_1$  at time 1 was removed at time 3. Table 5.1 summarizes the efficiency of different dynamic schemes, and their level of backward privacy.

In order to capture these notions, we will need some of the leakage components defined in Section 3.2.2, such as  $\text{up}$ ,  $\text{TimeDB}$ , or  $\text{DelHist}$ . With these tools, we can formally define our four notions of backward privacy.

**Table 5.1** – Comparison of different dynamic SSE schemes. The RT column stands for the number of round-trips in the search protocol. BP stands for backward privacy. Note that all the schemes presented in this table, except the first one ( $\Pi^{\text{dyn}}$ ), are forward private. We denote different levels of backward privacy with I, II, and III, where I is the strongest level (see Section 5.1 for details). The notation  $\tilde{\mathcal{O}}$  hides polylog factors. The client’s storage is given as the number of entries stored by the client. As such, each entry is of size  $\log D_{\max}$ , where  $D_{\max}$  is the maximum number of documents supported by the scheme.

Scheme	Computation		Communication			Client Storage	BP
	Search	Update	Search	RT	Update		
$\Pi^{\text{dyn}}$ [CJJ+14]	$\mathcal{O}(a_w)$	$\mathcal{O}(1)$	$\mathcal{O}(n_w)$	1	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
SPS [SPS14]	$\mathcal{O}\left(\min\left\{a_w + \log N, n_w \log^3 N\right\}\right)$	$\mathcal{O}(\log^2 N)$	$\mathcal{O}(n_w + \log N)$	1	$\mathcal{O}(\log N)$	$\mathcal{O}(N^\alpha)$	-
TWORAM [GMP16]	$\tilde{\mathcal{O}}(n_w \log N + \log^3 N)$	$\tilde{\mathcal{O}}(\log^3 N)$	$\tilde{\mathcal{O}}(n_w \log N + \log^3 N)$	2	$\tilde{\mathcal{O}}(\log^3 N)$	$\mathcal{O}(1)$	-
ARX [PBP16]	$\mathcal{O}(a_w)$	$\mathcal{O}(\log a_w)$	$\mathcal{O}(n_w + \log a_w)$	1	$\mathcal{O}(1)$	$\mathcal{O}(K)$	-
$\Sigma\phi\phi\phi$ § 4.5	$\mathcal{O}(a_w)$	$\mathcal{O}(1)$	$\mathcal{O}(n_w)$	1	$\mathcal{O}(1)$	$\mathcal{O}(K)$	-
Diana § 4.6	$\mathcal{O}(a_w)$	$\mathcal{O}(\log a_w)$	$\mathcal{O}(n_w + \log a_w)$	1	$\mathcal{O}(1)$	$\mathcal{O}(K)$	-
Fides § 5.2.1	$\mathcal{O}(a_w)$	$\mathcal{O}(1)$	$\mathcal{O}(a_w)$	2	$\mathcal{O}(1)$	$\mathcal{O}(K)$	II
Moneta § 5.2.2	$\tilde{\mathcal{O}}(a_w \log N + \log^3 N)$	$\tilde{\mathcal{O}}(\log^2 N)$	$\tilde{\mathcal{O}}(a_w \log N + \log^3 N)$	3	$\tilde{\mathcal{O}}(\log^3 N)$	$\mathcal{O}(1)$	I
Diana <sub>del</sub> § 5.3	$\mathcal{O}(a_w)$	$\mathcal{O}(\log a_w)$	$\mathcal{O}(n_w + d_w \log a_w)$	2	$\mathcal{O}(1)$	$\mathcal{O}(K)$	III
Janus § 5.4	$\mathcal{O}(n_w \cdot d_w)$	$\mathcal{O}(1)$	$\mathcal{O}(n_w)$	1	$\mathcal{O}(1)$	$\mathcal{O}(K)$	III

**Definition 5.1** (Backward Privacy). An  $\mathcal{L}$ -adaptively-secure SSE scheme is strongly backward-private if and only if the search and update leakage functions  $\mathcal{L}^{\text{Srch}}$ ,  $\mathcal{L}^{\text{Updt}}$  can be written as:

$$\begin{aligned}\mathcal{L}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \mathcal{L}'(\text{op}) \\ \mathcal{L}^{\text{Srch}}(w) &= \mathcal{L}''(\text{DB}(w)),\end{aligned}$$

where  $\mathcal{L}'$  and  $\mathcal{L}''$  are stateless.

An  $\mathcal{L}$ -adaptively-secure SSE scheme is insertion pattern revealing backward-private if and only if the search and update leakage functions  $\mathcal{L}^{\text{Srch}}$ ,  $\mathcal{L}^{\text{Updt}}$  can be written as:

$$\begin{aligned}\mathcal{L}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \mathcal{L}'(\text{op}) \\ \mathcal{L}^{\text{Srch}}(w) &= \mathcal{L}''(\text{TimeDB}(w), a_w),\end{aligned}$$

where  $\mathcal{L}'$  and  $\mathcal{L}''$  are stateless.

An  $\mathcal{L}$ -adaptively-secure SSE scheme is update pattern revealing backward-private if and only if the search and update leakage functions  $\mathcal{L}^{\text{Srch}}$ ,  $\mathcal{L}^{\text{Updt}}$  can be written as:

$$\begin{aligned}\mathcal{L}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \mathcal{L}'(\text{op}, w) \\ \mathcal{L}^{\text{Srch}}(w) &= \mathcal{L}''(\text{TimeDB}(w), \text{Updates}(w)),\end{aligned}$$

where  $\mathcal{L}'$  and  $\mathcal{L}''$  are stateless.

An  $\mathcal{L}$ -adaptively-secure SSE scheme is weakly backward-private if and only if the search and update leakage functions  $\mathcal{L}^{\text{Srch}}$ ,  $\mathcal{L}^{\text{Updt}}$  can be written as:

$$\begin{aligned}\mathcal{L}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \mathcal{L}'(\text{op}, w) \\ \mathcal{L}^{\text{Srch}}(w) &= \mathcal{L}''(\text{TimeDB}(w), \text{DelHist}(w)),\end{aligned}$$

where  $\mathcal{L}'$  and  $\mathcal{L}''$  are stateless.

We can clearly see that backward privacy with insertion pattern implies update pattern revealing backward privacy, which itself implies weak backward privacy. Also observe that an insertion pattern revealing backward-private scheme has to be forward-private, and that if a scheme is both forward-private and weakly backward-private, then the leakage of update queries cannot depend on either the updated keyword (by definition of forward privacy) or the updated document index (by definition of weak backward privacy), so the leakage must be limited to the nature of the operation. This will indeed be the case for all schemes considered in this thesis.

Finally, a strongly backward-private scheme has to hide the insertion pattern and can only be instantiated using oblivious RAM. We will not try to achieve this level of security in the following sections.

## 5.2 A Generic Two Round-trips Backward-Private Scheme

In this section, we show how to build a simple backward-private SSE scheme  $B(\Sigma)$  starting from an arbitrary SSE scheme  $\Sigma$ . We start with a basic solution for clarity, then improve on it.

We alter  $\Sigma$  as follows. Instead of storing a document index  $\text{ind}$ , the client uploads a ciphertext  $E_{K_w}(\text{ind}, \text{op})$ , where  $E_{K_w}$  is a secret-key encryption scheme and  $\text{op} \in \{\text{add}, \text{del}\}$ . The key  $K_w$  is specific to keyword  $w$  and is chosen by the client. The server sees only the resulting ciphertexts as



$K_w$ 's are never revealed to it. The scheme  $\Sigma$  otherwise runs as normal. In particular, Search queries return the set of matching encrypted document indices  $E_{K_w}(\text{ind}, \text{op})$ . The client can then decrypt this set, remove deleted indices, and obtain the final set of document indices matching  $w$ .

A description of  $B(\Sigma)$  is provided in Algorithm 5.9. Letting  $\mathcal{I}$  denote the set of document indices, we assume  $\mathcal{I} \times \{\text{add}, \text{del}\}$  embeds into the plaintext space of  $E_K$ , and we use the ciphertext space of  $E_K$  as the set of document indices for  $\Sigma$ . Note that  $\Sigma$  only needs to support add queries. The scheme  $B(\Sigma)$  achieves update pattern revealing backward privacy, as  $\Sigma$  can leak any information about the modified keyword during updates, and some access pattern information during search. However, if  $\Sigma$  does not reveal any information about the past updates (*i.e.*, if  $\Sigma$  does not leak  $\text{UpHist}(w)$  but only  $\text{DB}(w)$ ), we can show that  $B(\Sigma)$  guarantees backward-privacy with insertion pattern leakage. Unfortunately, the only dynamic schemes which do not reveal  $\text{UpHist}(w)$  are based on ORAM, such as TWORAM [GMP16].

---

**Algorithm 5.9** Generic backward-private scheme  $B(\Sigma)$  where  $\Sigma$  is an arbitrary SSE scheme and  $F$  is a PRF.

---

Setup( $DB$ ) :

- 1:  $\Sigma.\text{Setup}(DB), K_\Sigma \xleftarrow{\$} \{0, 1\}^\lambda$

Search( $K_\Sigma, w, \sigma$ ; EDB)

- 1: Client and Server run  $\Sigma.\text{Search}(w)$ , the client gets the list of results  $R$ .

*Client:*

- 2:  $K_w \leftarrow F(K_\Sigma, w)$
- 3: Decrypt  $R$  as  $(E_{K_w}(\text{ind}_1, \text{op}_1), \dots, E_{K_w}(\text{ind}_n, \text{op}_n))$
- 4: Return  $\{\text{ind} : \exists i, (\text{ind}_i, \text{op}_i) = (\text{ind}, \text{add}) \wedge \forall j > i, (\text{ind}_j, \text{op}_j) \neq (\text{ind}, \text{del})\}$

Update( $K_\Sigma, \text{add}, w, \text{ind}, \sigma$ ; EDB)

- 1: Client:  $K_w \leftarrow F(K_\Sigma, w)$
  - 2: Client and Server run  $\Sigma.\text{Update}(\text{add}, w, E_{K_w}(\text{ind}, \text{op}))$
- 

The  $B(\Sigma)$  scheme, as described so far, has two drawbacks. The first drawback is that the server does not learn document indices in the clear and, hence, cannot return the matching documents. This is fine for a result-hiding scheme. However, a common use case of SSE schemes is to return actual documents, which are stored separately in an encrypted form.  $B(\Sigma)$  can support this case with an additional round-trip as follows. After the client computes the result of a search query, he sends document indices in the clear to the server. The server is then able to send the documents to the client. Hence,  $B(\Sigma)$  is two round-trips protocol, assuming  $\Sigma$  requires a single round-trip for its queries.

The second drawback of  $B(\Sigma)$  is that deleted elements are never deleted on the server side. Moreover, since deleted elements are returned to the client on each search query, this also affects the communication cost and the amount of work necessary on the client side. We notice that this overhead can be avoided in the common scenario outlined above where the client sends cleartext document indices back to the server. In particular, it suffices for the client to send, together with the list of cleartext indices, an encryption of the same indices with a new key. Recall, that this list contains only the relevant indices with deleted elements removed by the client. Hence, the server can delete the old encrypted entries in the database and insert the updated ones. Essentially we are piggybacking a cleanup procedure on top of the Search protocol.

We denote a generic solution based on the above idea as  $B'(\Sigma)$  and describe it in Algorithm 5.10. In  $B'(\Sigma)$ , the client keeps track of the number of times each keyword  $w$  has been queried in table

---

**Algorithm 5.10** Improved generic backward-private scheme  $B'(\Sigma)$ .

---

**Setup**( $DB$ ) :

- 1:  $\mathbf{T}[w] \leftarrow 0$  for all  $w$ ,  $K_\Sigma \xleftarrow{\$} \{0, 1\}^\lambda$
- 2:  $DB' \leftarrow DB$  where keywords  $w$  are replaced by  $w||0$
- 3:  $\Sigma.\text{Setup}(DB')$

**Search**( $K_\Sigma, w, \sigma$ ; EDB)

- 1: Client:  $K_w \leftarrow H(K_\Sigma, w, \mathbf{T}[w])$
- 2: Client and Server run  $R \leftarrow \Sigma.\text{Search}(w||\mathbf{T}[w]) \triangleright$  Server can erase all retrieved elements from memory
- Client:
- 3: Decrypt  $R$  as  $(E_{K_w}(\text{ind}_1, \text{op}_1), \dots, E_{K_w}(\text{ind}_n, \text{op}_n))$
- 4:  $R' \leftarrow \{\text{ind} : \exists i, (\text{ind}_i, \text{op}_i) = (\text{ind}, \text{add}) \wedge \forall j > i, (\text{ind}_j, \text{op}_j) \neq (\text{ind}, \text{del})\}$
- 5: Send  $R'$  to Server
- 6:  $\mathbf{T}[w] \leftarrow \mathbf{T}[w] + 1$
- 7: **for all**  $\text{ind} \in R'$  **do**  $\triangleright$  In parallel
- 8:     Run  $\text{Update}(K_\Sigma, \text{add}, w, \text{ind}, \sigma$ ; EDB)
- 9: **end for**

**Update**( $K_\Sigma, \text{add}, w, \text{ind}, \sigma$ ; EDB)

- 1: Client:  $K_w \leftarrow H(K_\Sigma, w, \mathbf{T}[w])$
  - 2: Client and Server run  $\Sigma.\text{Update}(\text{add}, w||\mathbf{T}[w], E_{K_w}(\text{ind}, \text{op}))$
- 

$\mathbf{T}$ . Each time a search query is issued, results are re-encrypted using a fresh key derived from  $w$  and  $\mathbf{T}[w]$ . Keywords  $w$  in  $\Sigma$  are replaced by  $w||\mathbf{T}[w]$ , where  $||$  denotes concatenation. In line 7 of the algorithm, re-encrypted indices are sent as Update queries for the sake of having a generic solution. However, typical SSE schemes would allow all updates to be performed at once in a single round-trip. We also expect that concrete choices of  $\Sigma$  may allow further optimisations. For example, directly using a result-hiding scheme for  $\Sigma$  would avoid having to encrypt the  $(\text{ind}, \text{op})$  pairs before inserting them in  $\Sigma$ .

The scheme  $B'(\Sigma)$  is intuitively backward-private since the server learns document indices only after the client has removed deleted indices. Moreover, since document indices are re-encrypted after each search, it achieves the notion of update pattern revealing backward privacy in the sense of Definition 5.1.

### 5.2.1 Fides: A Baseline Forward and Backward Private SSE Scheme

We can now briefly describe Fides, the instantiation of  $B'$  using  $\Sigma\text{o}\phi\text{o}\varsigma$  (recall that  $\Sigma\text{o}\phi\text{o}\varsigma$  is forward-private, but not backward-private). Fides guarantees forward privacy and update pattern revealing backward privacy. The former is due to the underlying SSE scheme,  $\Sigma\text{o}\phi\text{o}\varsigma$ , being forward-private, while the latter is the result of the  $B'$  construction. The formal statement on Fides' security is given by Theorem 5.1.

**Theorem 5.1.** Define  $\mathcal{L}_{\text{Fides}}$  as:

$$\mathcal{L}_{\text{Fides}}^{\text{Srch}}(w) = (\text{DB}(w), \text{Updates}(w)),$$

$$\mathcal{L}_{\text{Fides}}^{\text{Updt}}(\text{op}, w, \text{ind}) = \perp.$$

Fides is  $\mathcal{L}_{\text{Fides}}$ -adaptively-secure. In particular, Fides is both forward-private and update pattern revealing backward-private.

Let us analyze Fides' performance. Recall that  $\Sigma\phi\phi\phi$  is optimal for search and updates in terms of computation and communication. In contrast, Fides takes two rounds during search and has  $\mathcal{O}(a_w)$  computation and communication complexity, where  $a_w$  is the total number of update entries matching  $w$ . The cost of  $\mathcal{O}(a_w)$  is the worst case scenario since this cost can be amortized over all search queries for  $w$ . Similar to  $\Sigma\phi\phi\phi$ , the updates in Fides are optimal (constant communication and computation).

Fides can be seen as a baseline for forward- and backward-private designs: it is simple to build, offers moderate computation overhead, and achieves a good level of security. In the next sections, we will propose schemes that avoid inefficiencies such as the additional round-trip and the high communication overhead at the cost of being only weakly backward-private.

### 5.2.2 Moneta: An (Almost) Strongly Backward Private Scheme

We note that  $B'(\Sigma)$  may achieve a stronger definition if one makes further assumptions on how updates are carried out in  $\Sigma$ . In particular, we name the  $B'(\text{TWORAM})$  instantiation Moneta. Moneta achieves backward privacy with insertion pattern, but at a very high computational and communication cost due to the use of TWORAM.

**Theorem 5.2.** Define  $\mathcal{L}_{\text{Moneta}}$  as:

$$\begin{aligned}\mathcal{L}_{\text{Moneta}}^{\text{Srch}}(w) &= (\text{DB}(w), a_w), \\ \mathcal{L}_{\text{Moneta}}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \perp.\end{aligned}$$

Moneta is  $\mathcal{L}_{\text{Moneta}}$ -adaptively-secure. In particular, Moneta is both forward-private and insertion pattern revealing backward-private.

Note that Moneta is almost strongly backward private but still leaks the number of insertions and deletions performed

Hence, Moneta is more secure than Fides, but has a worse computational complexity. Indeed, The search complexity is  $\tilde{\mathcal{O}}((a_w + d_w + n_w) \log N + \log^3 N) = \tilde{\mathcal{O}}(a_w \log N + \log^3 N)$ , and the update complexity is  $\tilde{\mathcal{O}}(\log^3 N)$ , which does not make it really practical.

## 5.3 Diana<sub>del</sub>: Backward Privacy from Range-Constrained and Puncturable PRFs

The FS-RCPRF construction, and its instantiation Diana (cf. Section 4.6), do not support deletions. Schemes of this type can be extended to support deletions by letting the client and the server maintain two instances of the construction, one for insertions and one for deletions. Then, during a search query, the server can compute the difference between the two result sets to compute the list of documents matching the query (i.e., without the deleted entries). This solution, however, is not backward-private as the server trivially learns the deleted entries. To this end, we propose FS-RCPRF<sub>del</sub>, which also uses two SE instances but exploits constrained PRFs to guarantee weak backward privacy.

The key idea behind FS-RCPRF<sub>del</sub> is to extend the set of constraints supported by the underlying constrained PRF used in Section 4.6.1. In order to support backward privacy, we make use of

constrained PRF  $\tilde{F}$  that is not only range-constrained (for forward privacy) but is *also* punctured on the deleted entries (for backward privacy). Hence, the constrained key of  $\tilde{F}$  enforces the predicate  $C_{c,x_1,\dots,x_n}(x) = 1$  if and only if  $x \in [0, c]$  and  $\forall i, x \neq x_i$ . The values  $x_1, \dots, x_n$  correspond to deleted entries that the server should not learn. Unfortunately, a naive implementation of  $\tilde{F}$  requires the client to store all deleted entries  $x_1, \dots, x_n$  since the order of deletions and insertions can be arbitrary. Our construction avoids this storage overhead on the client's side by letting the server store the deleted entries in an encrypted form.

We now combine the above ideas and describe  $\text{FS-RCPRF}_{\text{del}}$ . The client and the server maintain two forward-private SE instances: one for insertions and one for deletions. Every time the client wants to insert  $(w, \text{ind})$  with the counter  $c$ , it proceeds as in  $\text{FS-RCPRF}$  and inserts the pair in the first SE instance, as in Algorithm 4.6. In addition, it also pushes the pair  $(F'(K_w, (w, \text{ind})), \text{Enc}_{K'}(c))$ , where  $F'$  and  $\text{Enc}$  are a PRF and a CPA encryption scheme, to the server who stores these pairs in a map. In order to delete  $(w, \text{ind})$ , the client inserts the entry  $(w, F'(K_w, (w, \text{ind})))$  in the second SE instance. Then, during search query for  $w$ , the client proceeds as follows. It requests a search for  $w$  on the second SE instance (i.e., the one that stores deleted entries). As a result, the server gets the associated tags  $F'(K_w, (w, \text{ind}))$  for the deleted entries, uses them to retrieve encrypted  $x_i$ 's from the map, and sends them back to the client. The client then constrains the PRF using the  $x_i$ 's and uses it to run a search on the first SE instance. Note that this solution assumes that the same index  $\text{ind}$  is never reused: once the entry  $(w, \text{ind})$  has been deleted, it can no longer be re-added.

The above solution is not ideal as it requires an additional roundtrip with large communication from the server to the client. Also, it can only guarantee weak backward privacy, as the server learns when the deletions occurred.

Similar to  $\text{FS-RCPRF}$ , we instantiate  $\text{FS-RCPRF}_{\text{del}}$  with the GGM PRF and call the resulting scheme  $\text{Diana}_{\text{del}}$ . The constrained key, instead of consisting of the covering nodes of the full range as in Section 4.6.2, will be constructed as the set of nodes covering the ranges  $[0, x_1 - 1], [x_1 + 1, x_2 - 1], \dots, [x_n + 1, c]$  (assuming that  $x_i$ 's are in increasing order). We indeed combine a range-constrained PRF (constrained on the range  $[0, c]$ ) and a punctured PRF (punctured on the points  $x_1, \dots, x_n$ ). This approach will result in large keys when the number of deletions is large: the number of tree nodes to be sent will be in the order of  $d_w \cdot \log(n_w/d_w)$ . (assuming uniformly distributed deletions).

## 5.4 Janus: Weak Backward Privacy from Puncturable Encryption

The solutions presented in Section 5.3 suffer from high inefficiencies, by requiring either client storage linear in the number of deletions, or multiple round-trips with high communication complexity. In this section, we show how to achieve (weak) backward security in a single round-trip, using puncturable encryption with incremental punctures.

### 5.4.1 Puncturable Encryption

A puncturable encryption (PE) scheme is a public-key encryption scheme that allows for *puncturing* the secret key to prevent the decryption of some messages. More precisely, for such schemes, the plaintexts are encrypted and attached to a *tag*, and the secret key is punctured on a set of tags so that decryption of ciphertexts attached to those tags is impossible. Puncturable encryption has been introduced by Green and Miers as a way to achieve forward security in an asynchronous setting [GM15]. We adopt the same formalism and definitions, except we fix the number of tags per message to 1.

A puncturable encryption scheme PPKE with message space  $\mathcal{M}$  and tag space  $\mathcal{T}$  is a triple of algorithms (KeyGen, Encrypt, Puncture, Decrypt) with the following syntax:

- KeyGen( $1^\lambda$ ) outputs a public key PK and an initial secret key SK<sub>0</sub>.
- Encrypt(PK,  $M, t$ ) outputs the encryption  $CT$  of  $M \in \mathcal{M}$  attached to the tag  $t \in \mathcal{T}$ .
- Puncture(SK <sub>$i$</sub> ,  $t$ ) outputs a new secret key SK <sub>$i+1$</sub>  able to decrypt any ciphertext SK <sub>$i$</sub>  can decrypt, except for ciphertexts encrypted with the tag  $t$ .
- Decrypt(SK <sub>$i$</sub> ,  $CT, t$ ) outputs a plaintext  $M$  or  $\perp$  if the decryption fails.

Correctness is achieved if a plaintext  $M$  encrypted with tag  $t$  decrypts back to  $M$  when using the secret key punctured on any set of tags that does not contain  $t$ .

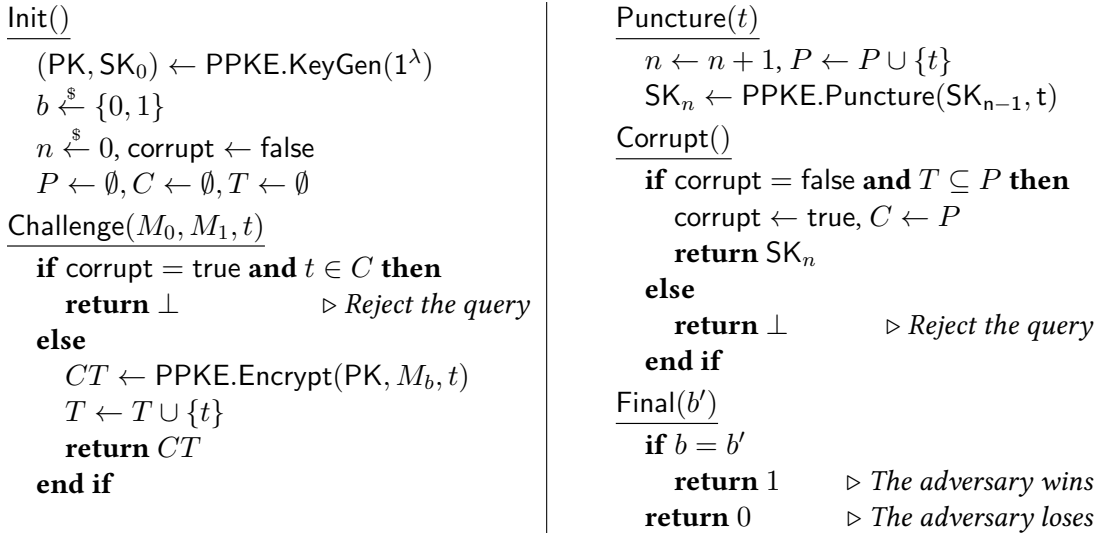
The IND-PUN-ATK security definitions – with  $\text{ATK} \in \{\text{CPA}, \text{CCA}\}$  – capture the security of puncturable encryption. We recall the IND-PUN-CPA game (we will not use CCA security in this work) in a simplified version.

**Definition 5.2** (Security of puncturable encryption). *Let PPKE be a puncturable encryption scheme. The advantage of an adversary  $A$  against PPKE in the IND-PUN-CPA security game is*

$$\text{Adv}_{\text{PPKE}, A}^{\text{pun-cpa}}(\lambda) = \left| \frac{1}{2} - \mathbb{P}[G_{\text{pun-cpa}}^A(\lambda) = 1] \right|$$

where  $G_{\text{pun-cpa}}$  is the game defined in Figure 5.1. We say that PPKE is IND-PUN-CPA secure if, for every polynomial-time adversary  $A$ ,

$$\text{Adv}_{\text{PPKE}, A}^{\text{pun-cpa}}(\lambda) \leq \text{negl}(\lambda).$$



**Figure 5.1** – Procedures of the  $G_{\text{pun-cpa}}$  security game. The game ensures that the adversary can get challenge ciphertexts only for tags on which the secret key has been punctured.

In the Janus construction, described in Section 5.4.3, we will encrypt the document indices using puncturable encryption, with tags that are pseudo-randomly generated from the document-keyword pairs. There will be a different key for each keyword, and when we want to delete an entry for a specific keyword, we will puncture the associated key on the tag derived from the document-keyword pair. Upon a search query, the client will give to the server the associated punctured secret key, with which he will only be able to decrypt non-deleted entries.

In this paper, we will use the Green-Miers puncturable encryption scheme [GM15], described in Algorithm 5.11. In [GM15], the authors show that their construction is IND-PUN-ATK secure if the Decisional Bilinear Diffie-Hellman (DBDH, cf. Section 2.2.2) hold in the groups  $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$  and  $\mathbb{G}_T$ .

---

**Algorithm 5.11** The Green-Miers puncturable encryption scheme, for message of  $m$  bits

---

**KeyGen**( $1^\lambda$ )

- 1: Choose a group  $\mathbb{G}$  of prime order  $p$  and generator  $g$ , and the hash functions  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ ,  $H' : \mathbb{G}_T \rightarrow \{0, 1\}^m$ .
- 2:  $\alpha, \beta, \gamma, r \xleftarrow{\$} \mathbb{Z}_p$ .  $g_1 \leftarrow g^\alpha$ ,  $g_2 \leftarrow g^\beta$ .
- 3: Define  $q(x) = \beta + \gamma \cdot x$  and  $V(x) = g^{q(x)}$ .
- 4: Let  $t_0$  be a distinguished tag, not to be used.
- 5: **return**  $\text{PK} = (g, g_1, g_2, g^{q(1)})$ ,  $\text{SK}_0 = [sk_0^{(1)} = g_2^{\alpha+r}, sk_0^{(2)} = V(H(t_0))^r, sk_0^{(3)} = g^r, t_0]$ .

**Encrypt**( $\text{PK}, M, t$ ) ( $M \in \{0, 1\}^m$ ,  $t \neq t_0$ )

- 1:  $s \xleftarrow{\$} \mathbb{Z}_p$
- 2: **return**  $(ct^{(1)} = M \oplus H'(e(g_1, g_2)^s), ct^{(2)} = g^s, ct^{(3)} = V(H(t))^s)$

**Puncture**( $\text{SK}_i, t$ ) ( $t \neq t_0$ )

- 1: Parse  $\text{SK}_i$  as  $[sk_0, sk_1, \dots, sk_i]$ , and  $sk_0$  as  $(sk_0^{(1)}, sk_0^{(2)}, sk_0^{(3)}, t_0)$
- 2:  $\lambda, r_0, r_1 \xleftarrow{\$} \mathbb{Z}_p$
- 3: Compute  $sk'_0 \leftarrow (sk_0^{(1)} \cdot g_2^{r_0 - \lambda'}, sk_0^{(2)} \cdot V(H(t_0))^{r_0}, sk_0^{(3)} \cdot g^{r_0}, t_0)$
- 4: Compute  $sk_{i+1} \leftarrow (g_2^{\lambda' + r_1}, V(H(t))^{r_1}, g^{r_1}, t)$
- 5: **return**  $[sk'_0, sk_1, \dots, sk_i, sk_{i+1}]$

**Decrypt**( $\text{SK}_i, CT, t$ )

- 1: Parse  $CT$  as  $(ct^{(1)}, ct^{(2)}, ct^{(3)})$ ,  $\text{SK}_i$  as  $[sk_0, sk_1, \dots, sk_i]$ .
  - 2: For  $j = 0, \dots, i$ , parse  $sk_j$  as  $(sk_j^{(1)}, sk_j^{(2)}, sk_j^{(3)}, t_j)$
  - 3: Compute  $\omega_j, \omega'_j$  s.t.  $\omega'_j \cdot q(H(t_j)) + \omega_j \cdot q(H(t)) = q(0) = \beta$
  - 4:  $Z_j \leftarrow \frac{e(sk_j^{(1)}, ct^{(2)})}{e(sk_j^{(3)}, ct^{(3)})^{\omega_j} \cdot e(sk_j^{(2)}, ct^{(2)})^{\omega'_j}}$
  - 5: **return**  $ct^{(1)} \oplus H' \left( \prod_{j=0}^i Z_j \right)$
- 

## 5.4.2 Incremental Puncture

The punctured keys will (often) grow with the number of punctures (or be very large), and it will be impractical to store them on the client side. To avoid this issue, we use an additional feature of the Green-Miers scheme, which we call incremental puncture.

In our setting, we will see that it is very handy to be able to express the Puncture algorithm as a function of a constant-sized fraction of the secret key. The secret key of the Green-Miers puncturable encryption scheme is, after  $n$  punctures,  $\text{SK}_n = (sk_0, sk_1, \dots, sk_n)$ , and the puncture algorithm is such that

$$\begin{aligned} \text{Puncture}(\text{SK}_n, t) &= (sk'_0, sk_1, \dots, sk_n, sk_{n+1}) \\ \text{where } (sk'_0, sk_{n+1}) &= \text{IncPuncture}(sk_0, t). \end{aligned}$$

By using this PE scheme, the client will only have to store the  $sk_0$  part of the secret key, and outsource the rest to the server. The client's storage will stay linear in the number of keywords, and most of the storage burden will still be born by the server.

### 5.4.3 The Janus Construction

Janus, similar to the constructions in Section 5.3, uses two forward-secure searchable encryption instances:  $\Sigma_{\text{add}}$  to store the newly inserted indices encrypted with the puncturable encryption scheme (the insertion instance), and  $\Sigma_{\text{del}}$  to store the punctured key elements (the deletion instance). There is a different puncturable encryption key for each keyword and the client stores the  $sk_0$  part of each key locally. During the search for  $w$ , the client sends the associated key part and runs the search protocol of the SE scheme for both instances. As a result, the server obtains the encrypted indices from the insertion instance and all the remaining key parts from the deletion instance. He will then be able to decrypt all the non-deleted (*i.e.* not punctured) indices.

Still, there is an important problem to tackle: once the secret key for  $w$  has been revealed to the server, it can no longer be used by the client to encrypt the index of the documents matching  $w$  that will be inserted in the future. As a consequence, we need to change the encryption key after every search. Yet, we do not need to re-encrypt the already revealed indices (*a.k.a.* the result indices) with the new key: the adversary already learned them, and, as the  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  schemes used in practice will leak the search pattern, he can keep track of the results over repeating search queries.

So, in the first version of our construction, the server will explicitly keep the results in a cache. This cache is also interesting from a performance point of view: each matching index will be decrypted at most once, and all the results from previous searches on a given keyword can be stored close to each other, increasing storage locality.

**Description of Janus.** Janus is described in Algorithm 5.12. It uses two response-revealing dynamic SSE schemes  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  supporting the insertion of atomic entries.  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  might be different for efficiency or security purposes, but in the proof, we will assume that they are both forward-private. Janus also uses a PRF  $F$  and an incremental puncturable encryption scheme PPKE.

The client stores locally a table containing for each keyword  $w$  the initial key share  $sk_0[w]$  of a puncturable encryption scheme (without loss of generality we can assume that this key share contains the public key). To insert a new entry  $(w, \text{ind})$ , the client encrypts it with the PE scheme with the key  $sk_0[w]$ , using a pseudo-random value  $F_{K_{\text{tag}}}(w, \text{ind})$  as a tag. He then inserts this ciphertext as a new entry matching  $w$  in  $\Sigma_{\text{add}}$ . To delete the entry  $(w, \text{ind})$ , the client computes the tag  $t = F_{K_{\text{tag}}}(w, \text{ind})$  and (incrementally) punctures  $sk_0[w]$  on this tag. He then updates the initial key share of  $w$  and pushes the new key element  $sk_t$  to the server by inserting the entry  $(w, sk_t)$  in  $\Sigma_{\text{del}}$ . Finally, to search, the client runs a search on  $w$  for both  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$ . The server now has access to the ciphertexts encrypting the inserted indices and to the key elements necessary to decrypt them. Note that he will only be able to decrypt the ciphertexts for which the key has not been punctured, *i.e.* the non deleted entries.

After a search query on  $w$ , the same encryption key cannot be used to encrypt new entries matching  $w$ : the server can reuse the old key to decrypt even the newly deleted entries since the key would not have been punctured on the corresponding tags. Janus avoids this by requiring the client to generate a new key for  $w$  after a search and encrypt new entries of  $w$  using this key. As discussed earlier, the server can keep the results of previous search queries and retrieve them the next time  $w$  is searched. This does not affect the security of the scheme since the server has already learnt earlier search results on  $w$ .

**Security of Janus.** Janus is a forward-private and weakly backward-private SSE scheme. The former comes directly from the forward security of  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$ . Let us consider backward security. The server has access to the decryption key of  $w$ 's entries only during the search query for  $w$ . Moreover, this key allows her to decrypt only the entries that have been added since the last search for  $w$  and have not yet been deleted. Hence, the deleted indices remain hidden. Note that weak backward security is the strongest definition we can achieve with Janus as the server can determine which of the inserted queries were later deleted as well as the timestamps of these events. Also note that Janus does not allow re-insertion of document/keyword pairs that were previously deleted.

**Theorem 5.3** (Adaptive Security of Janus). *If  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  are two  $L_{\text{FP}}$ -adaptively-secure SSE schemes, PPKE is IND-PUN-CPA secure, and  $F$  is a PRF, then Janus is  $\mathcal{L}_{wBS}$ -adaptively secure, with  $\mathcal{L}_{wBS} = (\mathcal{L}_{wBS}^{\text{Srch}}, \mathcal{L}_{wBS}^{\text{Updt}})$  defined as*

$$\begin{aligned}\mathcal{L}_{wBS}^{\text{Srch}}(w) &= (\text{sp}(w), \text{TimeDB}(w), \text{DelHist}(w)) \\ \mathcal{L}_{wBS}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \text{op}.\end{aligned}$$

Note that in this theorem,  $L_{\text{FP}}$  specifically refers to the leakage of a forward-secure scheme as defined in Definition 4.3. Namely,

$$\begin{aligned}L_{\text{FP}}^{\text{Srch}}(w) &= (\text{sp}(w), \text{Hist}(w)), \\ \text{and } L_{\text{FP}}^{\text{Updt}}(\text{op}, w, \text{ind}) &= \text{op},\end{aligned}$$

*Proof.* Again, we proceed by game hops.

**Game  $G_0$ .** This game is the real world SSE security game  $\text{SSER}_{\text{REAL}}$ .

$$\mathbb{P}[\text{SSER}_{\text{REAL}}^A_{\text{Janus}}(\lambda) = 1] = \mathbb{P}[G_0(1^\lambda) = 1]$$

**Game  $G_1$ .** In this game, we replace the calls to the PRF  $F$  with key  $K_S$  (resp.  $K_{\text{tag}}$ ) by picking new random outputs every time a previously unseen keyword (resp. document-keyword pair) is used. These strings are stored in a table to be reused every time  $F$  is again queried on  $w$  (resp.  $(w, \text{ind})$ ). Replacing  $F$  with key  $K_S$  this way induces a distinguishing advantage equal to the PRF distinguishing advantage for an adversary making  $W$  calls to  $F$ . Doing the same for  $F$  with key  $K_{\text{tag}}$  induces a distinguishing advantage equal to the PRF distinguishing advantage for an adversary making  $N$  calls to  $F$ . Hence, the adversarial distinguishing advantage between  $G_0$  and  $G_1$  is exactly twice the distinguishing advantage for the PRF  $F$ : we can build a reduction  $B_1$  making at most  $N$  calls on  $F$  such that

$$\mathbb{P}[G_0(1^\lambda) = 1] - \mathbb{P}[G_1(1^\lambda) = 1] \leq 2 \cdot \text{Adv}_{F, B_1}^{\text{prf}}(\lambda).$$



---

**Algorithm 5.12** Janus: weakly backward-secure SSE.
 

---

**Setup()**

- 1:  $(\text{EDB}_{\text{add}}, K_{\text{add}}, \sigma_{\text{add}}) \leftarrow \Sigma_{\text{add}}.\text{Setup}()$
- 2:  $(\text{EDB}_{\text{del}}, K_{\text{del}}, \sigma_{\text{del}}) \leftarrow \Sigma_{\text{del}}.\text{Setup}()$
- 3:  $K_{\text{tag}}, K_S \leftarrow \{0, 1\}^\lambda, \mathbf{PSK}, \mathbf{SC}, \text{EDB}_{\text{cache}} \leftarrow \text{empty map}$
- 4: **return**  $((\text{EDB}_{\text{add}}, \text{EDB}_{\text{del}}, \text{EDB}_{\text{cache}}), (K_{\text{add}}, K_{\text{del}}, K_{\text{tag}}, K_S), (\sigma_{\text{add}}, \sigma_{\text{del}}, \mathbf{PSK}, \mathbf{SC}))$

**Search** $(K_\Sigma, w, \sigma; \text{EDB})$ 
*Client:*

- 1:  $i \leftarrow \mathbf{SC}[w]$ .
- 2: **if**  $i = \perp$
- 3:   **return**  $\emptyset$
- 4: Send  $sk_0 = \mathbf{PSK}[w]$  to the server.
- 5:  $\mathbf{PSK}[w] \leftarrow \text{PPKE.KeyGen}(1^\lambda), \mathbf{SC}[w] \leftarrow i + 1$ .
- 6: Send  $\text{tkn} \leftarrow F(K_S, w)$  to the server.

*Client (C) & Server (S):*

- 7: C and S run  $\Sigma_{\text{add}}.\text{Search}(K_{\text{add}}, w || i, \sigma_{\text{add}}; \text{EDB}_{\text{add}})$ .  
The server gets a list  $((ct_1, t_1^{\text{add}}), \dots, (ct_n, t_n^{\text{add}}))$  of ciphertexts and tags.
- 8: C and S run  $\Sigma_{\text{del}}.\text{Search}(K_{\text{del}}, w || i, \sigma_{\text{del}}; \text{EDB}_{\text{del}})$ .  
The server gets a list  $((sk_1, t_1^{\text{del}}), \dots, (sk_m, t_m^{\text{del}}))$  of key elements.
- 9: S decrypts the ciphertexts with  $\text{SK} = (sk_0, sk_1, \dots, sk_m)$ , and obtains the list  $\text{NewInd} = ((\text{ind}_1, t_1), \dots, (\text{ind}_\ell, t_\ell))$ .

*Server:*

- 10:  $\text{OldInd} \leftarrow \text{EDB}_{\text{cache}}[\text{tkn}]$
- 11: Remove from  $\text{OldInd}$  the indices whose tags are in  $\{t_j^{\text{del}}\}$ .
- 12:  $\text{Res} \leftarrow \text{OldInd} \cup \text{NewInd}, \text{EDB}_{\text{cache}}[\text{tkn}] \leftarrow \text{Res}$
- 13: **return**  $\text{Res}$

**Update** $(K_\Sigma, \text{add}, w, \text{ind}, \sigma; \text{EDB})$ 

- 1:  $t \leftarrow F_{K_{\text{tag}}}(w, \text{ind})$
  - 2:  $sk_0 \leftarrow \mathbf{PSK}[w], i \leftarrow \mathbf{SC}[w]$
  - 3: **if**  $sk_0 = \perp$  **then**
  - 4:    $sk_0 \leftarrow \text{PPKE.KeyGen}(1^\lambda), \mathbf{PSK}[w] \leftarrow sk_0$
  - 5:    $i \leftarrow 0, \mathbf{SC}[w] \leftarrow i$
  - 6: **end if**
  - 7: **if**  $\text{op} = \text{add}$  **then**
  - 8:    $ct \leftarrow \text{PPKE.Encrypt}(sk_0, \text{ind}, t)$
  - 9:   Run  $\Sigma_{\text{add}}.\text{Update}(K_{\text{add}}, \text{add}, w || i, (ct, t), \sigma_{\text{add}}; \text{EDB}_{\text{add}})$
  - 10: **else** ▷  $\text{op} = \text{del}$
  - 11:    $(sk'_0, sk_t) \leftarrow \text{PPKE.IncPuncture}(sk_0, t)$
  - 12:   Run  $\Sigma_{\text{del}}.\text{Update}(K_{\text{del}}, \text{add}, w || i, (sk_t, t), \sigma_{\text{del}}; \text{EDB}_{\text{del}})$
  - 13:    $\mathbf{PSK}[w] \leftarrow sk'_0$
  - 14: **end if**
- 

**Game  $G_2$ .** This game replaces real calls to  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  by calls to the simulators. Yet, to do so, the game needs to keep track of all the updates as they come: it can no longer rely on the server to store them. So  $G_2$  makes some bookkeeping during the updates, and postpones all encryptions and

**Algorithm 5.13** Game  $G_2$ .

Setup()	Update( $K_\Sigma, \text{op}, w, \text{ind}, \sigma; \text{EDB}$ )
1: $\text{EDB}_{\text{add}} \leftarrow S_{\text{add}}.\text{Setup}()$ 2: $\text{EDB}_{\text{del}} \leftarrow S_{\text{del}}.\text{Setup}()$ 3: $\text{Tags}, \text{Tokens}, \text{Updates} \leftarrow \text{empty map}$ 4: $u \leftarrow 0, s \leftarrow 0$ 5: $\text{SC}, \text{EDB}_{\text{cache}} \leftarrow \text{empty map}$ 6: <b>return</b> $((\text{EDB}_{\text{add}}, \text{EDB}_{\text{del}}, \text{EDB}_{\text{cache}}),$ $(\text{Tags}, \text{Tokens}), (u, \text{Updates}, \text{SC}))$	1: Append $(u, \text{op}, \text{ind})$ to $\text{Updates}[w]$ 2: Run $S_{\text{op}}.\text{Update}(\perp)$
Search( $K_\Sigma, w, \sigma; \text{EDB}$ )	
<i>Client:</i>	
1: $i \leftarrow \text{SC}[w]$ . 2: <b>if</b> $i = \perp$ 3: <b>return</b> $\emptyset$ 4: $sk_0 \leftarrow \text{PPKE.KeyGen}(1^\lambda)$ 5: $L_{\text{add}}, L_{\text{del}}$ initialized to empty lists. 6: <b>for all</b> $(u_j, \text{op}, \text{ind}_j) \in \text{Updates}[w]$ <b>do</b> 7: $t_j \leftarrow \text{Tags}[w, \text{ind}_j]$ 8: <b>if</b> $\text{op} = \text{add}$ <b>then</b> 9: $ct_j \leftarrow \text{PPKE.Encrypt}(sk_0, \text{ind}_j, t_j)$ 10:     Append $(u_j, (ct_j, t_j))$ to $L_{\text{add}}$ 11: <b>else</b> 12: $(sk_0, sk_j) \leftarrow \text{PPKE.IncPuncture}(sk_0, t_j)$ 13:     Append $(u_j, (sk_j, t_j))$ to $L_{\text{del}}$ 14: <b>end if</b> 15: <b>end for</b> 16: Send $sk_0$ to the server. 17: $\text{SC}[w] \leftarrow i + 1$ . 18: Send $\text{tkn} \leftarrow \text{Tokens}[w]$ to the server.	
<i>Client &amp; Server:</i>	
19: Run the simulator $S_{\text{add}}.\text{Search}(s, L_{\text{add}})$ . The server gets a list $((ct_1, t_1^{\text{add}}), \dots, (ct_n, t_n^{\text{add}}))$ of ciphertexts and tags. 20: Run the simulator $S_{\text{del}}.\text{Search}(s, L_{\text{del}})$ . The server gets a list $((sk_1, t_1^{\text{del}}), \dots, (sk_m, t_m^{\text{del}}))$ of key elements. 21: S decrypts the ciphertexts with $\text{SK} = (sk_0, sk_1, \dots, sk_m)$ , and obtains the list $\text{NewInd} = ((\text{ind}_1, t_1), \dots, (\text{ind}_\ell, t_\ell))$ .	
<i>Server:</i>	
22: $\text{OldInd} \leftarrow \text{EDB}_{\text{cache}}[\text{tkn}]$ 23: Remove from $\text{OldInd}$ the indices whose tags are in $\{t_j^{\text{del}}\}$ . 24: $\text{Res} \leftarrow \text{OldInd} \cup \text{NewInd}, \text{EDB}_{\text{cache}}[\text{tkn}] \leftarrow \text{Res}$ 25: <b>return</b> $\text{Res}$	

key punctures to the subsequent Search query. We are able to do this only because both  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  are forward-secure: the updates leak no information on their content.

$G_2$  is precisely described in Algorithm 5.13. One very important thing is the way the lists  $L_{\text{add}}$

and  $L_{\text{del}}$  are created and used.  $L_{\text{add}}$  contains the encryption of the result indices for the search query, with their associated tag, and the insertion timestamp  $u$ . Similarly  $L_{\text{del}}$  is the list of key elements, associated tags and deletion timestamp. As such,  $L_{\text{add}}$  (resp.  $L_{\text{del}}$ ) corresponds to the update history on  $w$  for the scheme  $\Sigma_{\text{add}}$  (resp.  $\Sigma_{\text{del}}$ ), and is used as such by the simulator  $S_{\text{add}}$  (resp.  $S_{\text{del}}$ ).

From this, we can easily bound the distinguishing advantage between  $G_1$  and  $G_2$ . There exist two polynomial type adversaries  $B_{\text{add}}$  and  $B_{\text{del}}$  against  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  respectively, making at most  $N$  insertions, and two associated simulators  $S_{\text{add}}$  and  $S_{\text{del}}$  such that

$$\mathbb{P}[G_1(1^\lambda) = 1] - \mathbb{P}[G_2(1^\lambda) = 1] \leq \text{Adv}_{\Sigma, S_{\text{add}}, L_{\text{FP}}, B_{\text{add}}}^{\text{SSE-sim}}(\lambda) + \text{Adv}_{\Sigma, S_{\text{del}}, L_{\text{FP}}, B_{\text{del}}}^{\text{SSE-sim}}(\lambda).$$

**Game  $G_3$ .** Game  $G_3$  replaces the indices of the deleted documents by 0 when encrypting with the puncturable encryption scheme. Because we do this only for ciphertexts with punctured tags, the IND-PUN-CPA security of PPKE tells us that  $G_3$  is indistinguishable from  $G_4$ . There exist a reduction  $B_3$  such that

$$\mathbb{P}[G_2(1^\lambda) = 1] - \mathbb{P}[G_3(1^\lambda) = 1] \leq \text{Adv}_{\text{PPKE}, B_3}^{\text{pun-cpa}}(\lambda).$$

---

**Algorithm 5.14** Game  $G_3$ . Only Search is modified from  $G_2$

---

Search( $K_\Sigma, w, \sigma$ ; EDB)

```

  Proceed as in  $G_2$  until line 5
6: for all  $(u_j, \text{op}, \text{ind}_j) \in \text{Updates}[w]$  do
7:    $t_j \leftarrow \text{Tags}[w, \text{ind}_j]$ 
8:   if  $\text{op} = \text{add}$  then
9:     if  $\exists u'$  s.t.  $(u', \text{del}, \text{ind}_j) \in \text{Updates}[w]$  then  $\triangleright$  This entry has been deleted.
10:       $ct_j \leftarrow \text{PPKE.Encrypt}(sk_0, 0, t_j)$ 
11:    else
12:       $ct_j \leftarrow \text{PPKE.Encrypt}(sk_0, \text{ind}_j, t_j)$ 
13:    end if
14:    Append  $(u_j, (ct_j, t_j))$  to  $L_{\text{add}}$ 
15:  else
16:     $(sk_0, sk_j) \leftarrow \text{PPKE.IncPuncture}(sk_0, t_j)$ 
17:    Append  $(u_j, (sk_j, t_j))$  to  $L_{\text{del}}$ 
18:  end if
19: end for
  Proceed as in  $G_2$  from line 16

```

---

**Game  $G_4$ .** Game  $G_4$  (cf. Algorithm 5.15) explicitly uses the Updates table to compute the leakage information TimeDB and DelHist. Then, it uses this information to construct the lists  $L_{\text{add}}$  and  $L_{\text{del}}$  that will be passed to the simulator. Also, note that the tags, previous generated and stored from the document-keyword pair, are now generated on the fly, and not stored anymore. We can do that because we supposed that every document index was added a most once and deleted at most once. Tags cannot repeat and do not have to be stored to ensure consistency.

$G_4$  is pure rewriting of  $G_3$ , and

$$\mathbb{P}[G_3(1^\lambda) = 1] - \mathbb{P}[G_4(1^\lambda) = 1] = 0.$$

---

**Algorithm 5.15** Game  $G_4$ . Only Search is modified from  $G_3$

---

Search( $K_\Sigma, w, \sigma$ ; EDB)

Proceed as in  $G_3$  until line 0

6: TimeDB, DelHist initialized to empty lists.

7: **for all**  $(u_j, \text{add}, \text{ind}_j) \in \text{Updates}[w]$  **do**

8:   **if**  $\exists u'$  s.t.  $(u', \text{del}, \text{ind}_j) \in \text{Updates}[w]$  **then**  $\triangleright$  This entry has been deleted.

9:     Append  $(u_j, u')$  to DelHist

10:   **else**

11:     Append  $(u_j, \text{ind}_j)$  to TimeDB

12:   **end if**

13: **end for**

14: **for all**  $(u_j^{\text{add}}, u_j^{\text{del}}) \in \text{DelHist}$  sorted by increasing  $u_j^{\text{del}}$  **do**

15:    $t_j \leftarrow \{0, 1\}^\lambda$

16:    $ct_j \leftarrow \text{PPKE.Encrypt}(sk_0, 0, t_j)$

17:   Append  $(u_j^{\text{add}}, (ct_j, t_j))$  to  $L_{\text{add}}$

18:    $(sk_0, sk_j) \leftarrow \text{PPKE.IncPuncture}(sk_0, t_j)$

19:   Append  $(u_j^{\text{del}}, (sk_j, t_j))$  to  $L_{\text{del}}$

20: **end for**

21: **for all**  $(u_j, \text{ind}_j) \in \text{TimeDB}$  **do**

22:    $t_j \leftarrow \{0, 1\}^\lambda$

23:    $ct_j \leftarrow \text{PPKE.Encrypt}(sk_0, \text{ind}_j, t_j)$

24:   Append  $(u_j, (ct_j, t_j))$  to  $L_{\text{add}}$

25: **end for**

Proceed as in  $G_3$  from line 19

---

**Simulator.** The last thing remaining to build a simulator for Janus from  $G_4$  is to replace the explicit use of  $w$  to generate the token  $\text{tkn}$ . This can trivially be done using the search pattern  $\text{sp}(w)$ : we replace  $w$  by  $\min \text{sp}(w)$ . Also,  $S$  directly uses the leakage  $\text{TimeDB}(w)$  and  $\text{DelHist}(w)$  given as input of Search to generate  $L_{\text{add}}$  and  $L_{\text{del}}$ , and thus no longer needs to keep track of the updates, as in  $G_4$  (the leakage function  $\mathcal{L}_{wBS}$  does that for him). Finally,

$$\mathbb{P}[G_4(1^\lambda) = 1] - \mathbb{P}[\text{SSEIDEAL}_{\text{Janus}, S, \mathcal{L}_{wBS}}^A(\lambda) = 1] = 0.$$

**Conclusion.** By combining all the contributions from all the games, there exists 4 adversaries  $B_1, B_{\text{add}}, B_{\text{del}}$ , and  $B_3$  such that

$$\begin{aligned} \text{Adv}_{\text{Janus}, S, \mathcal{L}, A}^{\text{SSE-sim}}(\lambda) &\leq 2\text{Adv}_{F, B_1}^{\text{prf}}(\lambda) + \text{Adv}_{\Sigma, S_{\text{add}}, L_{\text{FP}}, B_{\text{add}}}^{\text{SSE-sim}}(\lambda) \\ &\quad + \text{Adv}_{\Sigma, S_{\text{del}}, L_{\text{FP}}, B_{\text{del}}}^{\text{SSE-sim}}(\lambda) + \text{Adv}_{\text{PPKE}, B_3}^{\text{pun-cpa}}(\lambda). \end{aligned}$$

□

**Efficiency.** The computational and communication complexity of Janus can easily be derived from  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$ . In particular, it has the same complexity for insertion (resp. deletion) updates as  $\Sigma_{\text{add}}$  (resp.  $\Sigma_{\text{del}}$ ). To analyze search queries, let  $T_{\text{add}}(n)$  and  $T_{\text{del}}(n)$  be the computational complexities of the search protocols of  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$ , respectively, where  $n$  is the size of a result set. Then, Janus'

search complexity for a keyword  $w$  with  $a_w$  insertions,  $d_w$  deletions, and  $n_w = a_w - d_w$  non-deleted matching results, is  $T_{\text{add}}(a_w) + T_{\text{del}}(d_w) + \mathcal{O}(n_w \cdot d_w)$ . The last term comes from the fact that a decryption of the PE scheme has complexity linear in the number of punctures. When instantiated with Diana or  $\Sigma\phi\phi\phi$ , Janus thus has search complexity  $\mathcal{O}(a_w + d_w + n_w \cdot d_w) = \mathcal{O}(n_w \cdot d_w)$ .

In terms of communication for search queries, Janus also inherits from the complexity of  $\Sigma_{\text{add}}$ , and  $\Sigma_{\text{del}}$ . Let  $C_{\text{add}}(n)$  and  $C_{\text{del}}(n)$  be the communication complexities of search protocols of  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$ , respectively, for a keyword that was inserted  $n$  times. Then, the communication complexity  $C_{\text{Janus}}(a_w, d_w)$  for a keyword that was inserted  $a_w$  times and deleted  $d_w$  times is  $C_{\text{add}}(a_w) + C_{\text{del}}(d_w)$ . Also, the number of round-trips is the maximum number of round-trips between  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$ . Hence, when instantiated either with  $\Sigma\phi\phi\phi$  or Diana, Janus has single round-trip search and updates protocols. In the case of  $\Sigma\phi\phi\phi$ , the search communication complexity is optimal (constant), and for Diana, it is  $\mathcal{O}(\log(a_w) + \log(d_w))$ .

#### 5.4.4 Reducing the Storage Overhead

In practice, the storage overhead of Janus is quite high: the client needs to store 3 group elements (at least 256 bits each) for every keyword, while each ciphertext on the server side consists of the masked index, two group elements and the tag, and 3 group elements and a tag for each key share. To reduce the overhead at the client, we use a trick similar to the one used in  $\Sigma\phi\phi\phi$ : we pseudo-randomly generate the encryption scheme's parameters and key elements  $sk_i$  from a master key and the number of punctures done on the secret key. The client does not need to store the public key as he can directly encrypt the plaintext indices from the scheme's parameters (and this will actually be faster). As a result, the client has to store only the number of deleted entries for each  $w$ , which he does already if  $\Sigma_{\text{del}}$  is instantiated with Diana. This modification is described in detail in Algorithm 5.16.

A similar trick can be used to reduce the storage on the server side. Indeed, one of the three group elements stored for each entry is a random blinding element, which can be generated pseudo-randomly using a PRF applied on the keyword/document pair  $(w, \text{ind})$  to be encrypted. As the blinding element is part of the ciphertext, and as it is now a (deterministic) function of the pair  $(w, \text{ind})$ , the tag is now redundant and can be omitted. This modification is also described in Algorithm 5.16.

Note that this optimization removes the plausible deniability property of Janus: because all the key elements are pseudo-randomly generated, they can be easily reconstructed from the puncturable encryption scheme's parameters, which now have to be stored by the client (which was not the case with the non-optimized version).

#### 5.4.5 Security of Janus Against Weaker Adversaries

We showed that Janus protects against persistent adversaries (e.g. a malicious server) and guarantees both forward and backward privacy. In this section, we analyze its resistance against weaker adversaries. First, we consider a *snapshot adversary* who is able to see the encrypted database at one (or more) instant – e.g. in case of a disk theft or subpoena. Then, we consider the security of Janus against a *late-persistent* adversary that obtains control over the server sometime after the client has outsourced his data and, possibly executed some queries – e.g. in case of malware.

Janus, as is, does not protect against a snapshot adversary since the cached results are kept in plaintext on the server side. Beside trivially revealing the cached content, this can also lead to the recovery of some of the queries. This, in turn, can be used for leakage-abuse attacks in the manner

---

**Algorithm 5.16** Our adaptation of Green-Miers' scheme for pseudo-random generation of parameters and randomness from a master secret key  $K_w^{GM}$ . We suppose that the group  $\mathbb{G}$  and the functions  $H$  and  $H'$  are picked externally.  $F_p : \{0, 1\}^* \rightarrow \mathbb{Z}_p$  is a PRF.

---

ParamGen( $K_w^{GM}$ )

- 1:  $\alpha \leftarrow F_p(K_w^{GM}, \text{alpha}), \beta \leftarrow F_p(K_w^{GM}, \text{beta}), \gamma \leftarrow F_p(K_w^{GM}, \text{gamma}), r \leftarrow F_p(K_w^{GM}, \text{r0}||0)$ .
- 2:  $g_1 \leftarrow g^\alpha, g_2 \leftarrow g^\beta$ .
- 3: **return**  $(\alpha, \beta, \gamma, r, g_1, g_2)$ .

Encrypt( $K_w^{GM}, M, t$ ) ( $M \in \{0, 1\}^m, t \neq t_0$ )

- 1:  $(\alpha, \beta, \gamma, r, g_1, g_2) \leftarrow \text{ParamGen}(K_w^{GM})$ .
- 2:  $s \xleftarrow{\$} F(K_w^{GM}, \text{s}||w||\text{ind})$
- 3:  $f \leftarrow g^s, h \leftarrow H(t)$
- 4: **return**  $(ct^{(1)} = M \oplus H'(e(g_1, f^\beta)), ct^{(2)} = f, ct^{(3)} = f^{\beta+h\cdot\gamma})$

IncPuncture( $K_w^{GM}, i, t$ )

- 1:  $(\alpha, \beta, \gamma, r, g_1, g_2) \leftarrow \text{ParamGen}(K_w^{GM})$ .
- 2:  $h \leftarrow H(t)$
- 3:  $r_1 \leftarrow F_p(K_w^{GM}, \text{r1}||i)$
- 4:  $\ell_i \leftarrow F_p(K_w^{GM}, \text{1}||i), \ell_{i-1} \leftarrow F_p(K_w^{GM}, \text{1}||(i-1))$
- 5:  $f \leftarrow g^{r_1}$
- 6: **return**  $(g^{\beta\cdot(\ell_i-\ell_{i-1}+r_1)}, f^{\beta+h\cdot\gamma}, f, t)$

SK0Gen( $K_w^{GM}, i$ )

- 1:  $(\alpha, \beta, \gamma, r, g_1, g_2) \leftarrow \text{ParamGen}(K_w^{GM})$ .
  - 2:  $h_0 \leftarrow H(t_0)$
  - 3: **if**  $i = 0$  **then**
  - 4:    $f \leftarrow g^r$
  - 5:   **return**  $(g^{\beta\cdot(r+\alpha)}, f^{\beta+h_0\cdot\gamma}, f)$
  - 6: **else**
  - 7:    $r_0 \leftarrow F_p(K_w^{GM}, \text{r0}||i)$
  - 8:    $\ell_i \leftarrow F_p(K_w^{GM}, \text{1}||i)$
  - 9:    $f \leftarrow g^{r_0}$
  - 10:   **return**  $(g^{\beta\cdot(r_0-\ell_i)}, f^{\beta+h_0\cdot\gamma}, f)$
  - 11: **end if**
- 

of file injections attacks [ZKP16] adapted to using a single (or multiple) snapshot of EDB (and in particular of  $\text{EDB}_{\text{cache}}$ ).

To fix this problem, we propose to encrypt  $\text{EDB}_{\text{cache}}$  using a key that is not permanently stored at the server: the content of  $\text{EDB}_{\text{cache}}$  relevant to  $w$  is encrypted using a keyword-specific symmetric key  $K_w$ . To this end, we modify line 6 of Algorithm 5.12 to  $\text{tkn}||K_w \leftarrow F(K_S, w)$  where the client also sends  $K_w$  to the server. Then the server uses  $K_w$  to decrypt and re-encrypt  $\text{EDB}_{\text{cache}}$  as needed, using an IND-CPA-secure secret-key encryption scheme  $E_{K_w}$ . Once the Search query is processed, the server discards  $K_w$ ; in particular it must not be stored in EDB.

Unfortunately, encryption alone is not sufficient as the implementation of  $\text{EDB}_{\text{cache}}$  could leak additional information, such as the time of insertion/modification of data, or the size of previous, now discarded, values. To this end, we rely on *history-independent (HI) data structures* [NT01]

whose goal is to hide exactly this kind of side-channel information. Note that if  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$  are instantiated with existing forward-secure schemes (SPS [SPS14],  $\Sigma\phi\phi\phi$ , or Diana – cf. Chapter 4), history-independence is not an issue as the snapshot adversary learns at most the update leakage, reduced to the list  $(\text{op}_i)$  with  $\text{op}_i = \text{add}$  if the  $i$ -th update was an insertion, and  $\text{op}_i = \text{del}$  otherwise. Though HI data structures come with an additional overhead, the state-of-the-art constructions are practical [BS13].

The security of the above approach relies on cooperation from the server who is required to use encryption and HI structures for  $\text{EDB}_{\text{cache}}$  and erase  $K_w$  from memory once he finishes en/de-crypting  $\text{EDB}_{\text{cache}}$ . Note that snapshot attacks are essentially attacks against the server, more so than against the client: we are protecting from the attacker information learned by the server.

Despite the assumptions we have just outlined, it is clear that storing the cache in encrypted form is a vast improvement over storing this information in cleartext. It is also a cheap solution: symmetric encryption is extremely fast on modern processors, especially in the presence of specialized instructions such as AES-NI. Encrypting  $\text{EDB}_{\text{cache}}$  would not significantly impact performance, relative to the decryption of punctured encryption schemes, or running the two SSE schemes  $\Sigma_{\text{add}}$  and  $\Sigma_{\text{del}}$ .

Let us now consider Janus against *late-persistent* adversaries. In this case, we strive to obtain the following backward privacy: even if a deleted entry matched a search query processed *before* the corruption, it should be infeasible for the adversary to recover the associated document index. Symmetrically encrypting  $\text{EDB}_{\text{cache}}$ , as in the case of the snapshot adversary, is no longer sufficient as the encryption key  $K_w$  will eventually be revealed. Instead, we require that the server encrypts results with the PE scheme, using the public key for the newly generated secret key (line 5 in Algorithm 5.12).

#### 5.4.6 Performance of Janus

As Janus is a composition of any forward secure scheme and the adapted Green-Miers puncturable encryption scheme, here we focus on the performance of this scheme once tweaked to reduce the storage overhead.

For the bilinear maps, we used a Type-3 pairing (cf. [GPS06]) on Barreto-Naehrig curves [BN06]. We modified Miers’ implementation of the Green-Miers PE scheme of libforwardsec [Mie15], which is itself based on the RELIC pairing library [AG], to fit our usage. The machine used for the experiments is the same as the one of Section 4.7.2: an Intel Core i7 4790K 4.00 GHz CPU (with 8 logical cores) with 16 GB of RAM. Note though that the experiments here were run using a single core.

**Table 5.2** – Performance of the puncturable encryption scheme used in Janus. Means are taken over 400 iterations.

Encrypt	IncPuncture	SK0Gen	Decrypt ( $d$ punctures)
1.699 ms	1.386 ms	1.396 ms	$(d + 1) \times 2.345$ ms

We end up having 74-byte ciphertexts (for 8-byte indices), and 200 byte key shares. The computational performance of the scheme is given in Table 5.2. SK0Gen is the procedure used to generate the first key share  $sk_0$  of the punctured secret key from the number of punctures. Note that these are single-core timings. While encryption, puncture and first key share generation are fast enough to yield a reasonably practical scheme, decryption does not scale well as the number of punctures

grows. In particular, Janus would not support more than a few hundreds deletions per keyword in practice, for both computational and storage overhead reasons.

Designing puncturable encryption with smaller keys or better computational efficiency is an open problem, and Janus would immediately benefit from any improvement in this area.

## References

- [AG] D. F. Aranha and C. P. L. Gouvêa. *RELIC is an Efficient Library for Cryptography*. [Link](#). (Cit. on p. 115).
- [BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. *Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives*. In: *ACM CCS 17*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1465–1482 (cit. on pp. viii, ix, 10, 13, 65, 97).
- [BN06] Paulo S. L. M. Barreto and Michael Naehrig. *Pairing-Friendly Elliptic Curves of Prime Order*. In: *SAC 2005*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. LNCS. Springer, Heidelberg, Aug. 2006, pp. 319–331 (cit. on p. 115).
- [BS13] Sumeet Bajaj and Radu Sion. *HIFS: history independence for file systems*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 1285–1296 (cit. on p. 115).
- [CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. 10, 49, 56, 73, 83, 99, 129, 163, 170, 171).
- [GM15] Matthew D. Green and Ian Miers. *Forward Secure Asynchronous Messaging from Puncturable Encryption*. In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 305–320 (cit. on pp. 104, 106).
- [GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. *TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption*. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563–592 (cit. on pp. 7, 8, 45, 69, 73, 99, 101, 161, 163, 170).
- [GPS06] S.D. Galbraith, K.G. Paterson, and N.P. Smart. *Pairings for Cryptographers*. Cryptology ePrint Archive, Report 2006/165. <http://eprint.iacr.org/2006/165>. 2006 (cit. on pp. 29, 115).
- [Mie15] Ian Miers. *Libforwardsec. Forward secure encryption for asynchronous messaging*. 2015. [Link](#). (Cit. on p. 115).
- [NT01] Moni Naor and Vanessa Teague. *Anti-persistence: History independent data structures*. In: *33rd ACM STOC*. ACM Press, July 2001, pp. 492–501 (cit. on p. 114).
- [PBP16] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. *Arx: A Strongly Encrypted Database System*. Cryptology ePrint Archive, Report 2016/591. <http://eprint.iacr.org/2016/591>. 2016 (cit. on pp. 9, 12, 87, 99).



- [SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. [viii–x](#), [10](#), [13](#), [45](#), [59](#), [67](#), [71](#), [73](#), [81](#), [97–99](#), [115](#), [119](#), [122](#), [130–133](#), [141](#), [147](#), [163](#), [170](#)).
- [ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. *All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption*. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 2016, pp. 707–720 (cit. on pp. [10](#), [12](#), [60](#), [66](#), [76](#), [114](#), [152](#), [153](#), [156](#), [157](#)).

Love all, trust a few, do wrong to none.

*All's Well That Ends Well* – WILLIAM  
SHAKESPEARE

# Verifiable Searchable Encryption

# 6



ALL THE SCHEMES STUDIED IN THE PREVIOUS CHAPTERS, are secure in the sense of confidentiality, when the server is honest-but-curious. On the opposite, malicious servers could easily modify the results of a query, *e.g.* for censorship purposes, and could even use the fact that the execution of the protocols is not verified to also break the confidentiality of the SSE construction.

In this chapter, we will study and design efficient SSE schemes provably secure against *malicious* servers. We will start by giving lower bounds on the complexity of such verifiable SSE schemes. Then, we will construct generic solutions matching these bounds using efficient verifiable data structures. We will quickly see how to adapt  $\Sigma\phi\phi\phi$ , Diana and Janus in order to turn them into verifiable schemes. Finally, we will modify the SPS forward private scheme [SPS14] to make it provably secure against active adversaries, without increasing the computational complexity of the original scheme.

These contributions, authored with Fouque and Pointcheval, appeared in [BFP16].

## Contents

---

<b>6.1</b>	<b>A Lower Bound on Verifiable Searchable Encryption</b>	<b>120</b>
6.1.1	Memory Checking	120
6.1.2	A General Lower Bound on Verifiable SSE	120
6.1.3	Lower Bound for Practical Constructions	121
<b>6.2</b>	<b>Tools for Constructing Verifiable Dynamic SSE schemes</b>	<b>122</b>
6.2.1	Verifiable Hash Tables	123
6.2.2	Static Verifiable Hash Tables	125
6.2.3	Incremental Hashing	126
<b>6.3</b>	<b>A Generic and Optimal Construction</b>	<b>127</b>
<b>6.4</b>	<b>Verifying <math>\Sigma\phi\phi\phi</math>, Diana, and Janus</b>	<b>129</b>
<b>6.5</b>	<b>Verifying SPS</b>	<b>130</b>
6.5.1	Remembering SPS	130
6.5.2	Quick Cryptanalysis of SPS	131
6.5.3	Verifiable SPS: Basic Construction	132
6.5.4	Sublinear Construction	134
6.5.5	Soundness proof of Verif-SPS	142
6.5.6	Complexity	147

---

## 6.1 A Lower Bound on Verifiable Searchable Encryption

Before giving some constructions of verifiable SSE, we want to show that protection against active adversaries has an inherent cost, and to lower bound this cost. For semi-honest adversaries, lower bounds are trivial: search cannot be done in less than  $\Omega(m)$  where  $m$  is the number of results for a query, and update has to run in  $\Omega(1)$  per modified document/keyword pair.

For malicious adversaries, the result is not as straightforward. Fortunately, we can rely on the literature on authenticated data structures (namely authenticated hash tables) [TT05; PT08; PTT09] and memory checking [BEG+91; NR05; DNRV09] to have a better insight into this question. Actually, in this section, we show how to reduce memory checking to verifiable SSE, and, using the lower bound result of [DNRV09], give a general computational lower bound on the Search and Update algorithms of *any* SSE scheme secure against malicious adversaries. Then, based on [TT05], we argue that, if we only use symmetric primitives for verification, we cannot actually hope for less than  $\Omega(n_w + \log K)$  for Search and  $\Omega(\log K)$  for Update.

### 6.1.1 Memory Checking

Memory checking is the problem of outsourcing memory to an untrusted party while ensuring authenticity and using limited trusted local storage. A memory checker  $\mathcal{C}$  is a probabilistic algorithm that receives from the user read and write queries, and, by making its own requests to the untrusted remote memory, and accessing a small private memory, ensures that the original queries were either answered correctly, or that a fault was reported.

The formal definition of a memory checker can be found in [DNRV09]. In particular, the authors define  $\mathcal{C}$  as a  $(\Sigma, n, q, s)$ -checker as a checker that can be used to store a database of  $n$  values with query complexity  $q$  (the average number of remote memory accesses necessary for each memory query processed by the memory checker) and local memory complexity  $s$ , where the secret and public memory are over the alphabet  $\Sigma$ .

The authors of this work prove a lower bound on  $q$ , assuming that the private memory is not ‘large’:  $s$  is  $\mathcal{O}(n^\alpha)$ , with  $0 \leq \alpha < 1$ . The actual theorem is given as follows:

**Theorem 6.1** (Theorem 3.1 of [DNRV09]). *Let  $\mathcal{C}$  be a  $(\Sigma, n, q, s)$  memory checker with  $s \leq n^{1-\varepsilon}$  for some  $\varepsilon > 0$  and  $|\Sigma| \leq n^{\text{poly}(\log n)}$ . It must be that  $q = \Omega\left(\frac{\log n}{\log \log n}\right)$ .*

### 6.1.2 A General Lower Bound on Verifiable SSE

The problem of verifiable SSE (VSSE) is somewhat similar to the one of memory checking: SSE stores document indices as values, and keywords replace memory checkers’ addresses. However, we can identify two important caveats:

- SSE schemes must be able to store a variable number of indices (*i.e.* keywords);
- in SSE, each keyword can match zero, one, or more documents, instead of having one memory address associated to exactly one value in the case of memory checkers;
- contrary to memory checkers, usual SSE schemes do not have an interface to replace a value/document by another one. It can only remove or add a keyword/document pair.

However, we can port the lower bound for memory checkers to VSSE, as stated in Theorem 6.2.

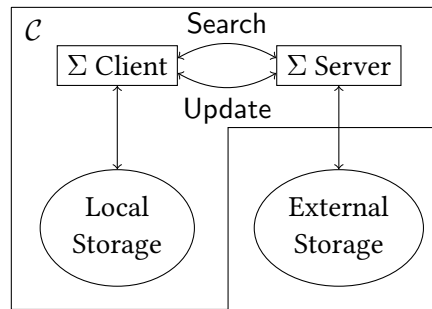
**Theorem 6.2** (Computational lower bound on verifiable SSE). *Let  $\Sigma$  be a VSSE scheme with client memory of size  $s \leq K^{1-\varepsilon}$  for some  $\varepsilon > 0$ . Then, either Search queries have computational complexity  $\Omega(\max(\frac{\log K}{\log \log K}, m))$  or Update queries have computational complexity  $\Omega(\frac{\log K}{\log \log K})$ .*

The proof will work by writing a reduction from memory checkers to verifiable SSE, and conclude using the lower bounds on memory checkers from Theorem 6.1 will also transfer to VSSE.

*Proof.* Search queries cannot be computed in less than  $\mathcal{O}(m)$  operations, trivially. So in the following, we will just show that they cannot either be executed in less than  $\mathcal{O}(\frac{\log K}{\log \log K})$ .

Let us first consider the memory checker  $\mathcal{C}$  built over an SSE scheme  $\Sigma$  as follows. In order to read index  $i$ ,  $\mathcal{C}$  calls  $\Sigma.\text{Search}(i)$  and returns the result. To write value  $v$  at index  $i$ ,  $\mathcal{C}$  first calls  $\Sigma.\text{Search}(i)$ , gets a single value  $v'$ , runs  $\Sigma.\text{Update}(\text{del}, i, v')$  and then  $\Sigma.\text{Update}(\text{add}, i, v)$ . In this execution,  $\Sigma$  stores exactly one value (document) per index (keyword), so the Search calls will only return a single entry.

One must notice that  $\mathcal{C}$ 's untrusted external memory is the memory used by the SSE server, and the external memory accesses  $\mathcal{C}$  makes are the ones made by the server side of the SSE. Similarly,  $\mathcal{C}$ 's trusted local memory is the one used by the SSE client, and so, the size  $s$  of private memory  $\mathcal{C}$  uses is  $s$ , the size of private memory of  $\Sigma$ , which we set to be less than  $K^{1-\varepsilon}$  for  $\varepsilon > 0$ . Figure 6.1 gives a simple graphic representation of the reduction and its limits.



**Figure 6.1** – Representation of the reduction from memory checkers to verifiable SSE. Everything in the enclosing box is comprised in  $\mathcal{C}$ .

If both the client and server-side parts of Search and Update had an (amortized) query complexity  $o(\frac{\log K}{\log \log K})$ , the query complexity of  $\mathcal{C}$  would be  $o(\frac{\log K}{\log \log K})$ , and thus  $\mathcal{C}$  would break the lower bound of Theorem 6.1. Also, without loss of generality, we can suppose that  $\Sigma$  uses all the data retrieved from the external memory. Hence, one of Search or Update has complexity at least  $\Omega(\frac{\log K}{\log \log K})$ .  $\square$

We emphasize that this result bounds the minimal complexity of the costliest operation between Search and Update and does not imply that *both* operations complexity is lower bounded by  $\Omega(\frac{\log K}{\log \log K})$ . Also, our result does not provide a lower bound on the communication complexity: we will actually show in Section 6.3 that we can achieve SSE verification with constant communication overhead.

### 6.1.3 Lower Bound for Practical Constructions

Theorem 6.2 gives us a general bound for generic searchable encryption. However, in this chapter, we focus more on *symmetric* searchable encryption schemes that (mostly) use symmetric primitives like

hash functions or block ciphers. More exactly, all the existing schemes use deterministic encryption to perform the search and update protocols (deterministically too), and we might want to use similar techniques for verification.

One technique we can rely on makes use of cryptographic hash functions for verification. The optimality of such constructions has been studied by Tamassia and Triandopoulos [TT05], who showed that one cannot do better than having  $\Omega(\log \frac{n}{k})$  verification and update computational complexity to authenticate a dictionary with  $n$  entries through hashing using memory  $\Theta(k)$  [TT05, Theorem 6]. They also show, that in this case, the communication overhead is also  $\Omega(\log \frac{n}{k})$ . Using the same reduction as in Theorem 6.2, we can show that if we rely on hashing for verification purpose in SSE, both Search and Update protocols cannot be run in time less than  $\Omega(\log \frac{K}{|\sigma|})$  time for a client side storage of size  $\Theta(|\sigma|)$ . Taking into account the search operation's lower bound for general SSE, we actually have that the minimal search complexity is  $\Omega(\max(m, \log \frac{K}{|\sigma|}))$ .

**Theorem 6.3.** *Let  $\Sigma$  be a VSSE scheme with client memory of size  $s$ , using hash functions for verification. Then, either Search queries have computational complexity  $\Omega(\max(\log \frac{K}{|\sigma|}, m))$  or Update queries have computational complexity  $\Omega(\log \frac{K}{|\sigma|})$ .*

Another well-known technique in authenticated data structures is cryptographic accumulators. In particular, it has been used in several contributions for verifiable hash tables [CL02; STY01]. The most efficient construction [PTT09] performs constant (expected) query time (for both proof generation and verification). However, the expected update time on the server side is  $\mathcal{O}(n^\epsilon)$  for a fixed  $0 < \epsilon < 1$  (it is constant on the client side), and we cannot reach the general lower bound of Section 6.1.2 using cryptographic accumulators. For practical considerations, we must also consider the fact that the cryptographic operations needed for accumulators (exponentiation of large integers or bilinear maps) are a lot more computationally expensive than hashing, and despite a better asymptotical complexity, they might not compare favorably with hashing-based techniques.

## 6.2 Tools for Constructing Verifiable Dynamic SSE schemes

When verifying the results of a search query, one has to check two points: first that every result matches the query, and then that every matching result has been returned. This crucial fact was already emphasized in [SPS14].

If the first point is relatively easy to ensure for static databases, things become more complicated when considering dynamic databases, especially databases supporting deletions. We must indeed prevent *replay attacks*, where the server returns a result that *used to* belong to the database but that no longer does (the document-keyword pair was deleted). If using MACs over the document-keyword pairs would have been enough for static SSE or even insertion-only SSE, this is no longer secure on its own for the completely dynamic case, when the SSE supports both insertions and deletions. Somehow, the client has to store some kind of digest of the outsourced database. For efficiency, we want this digest to be much smaller than the database.

The second point, is no less important, e.g. when a search query has no matching result. In this case, how can we prove that there are actually no matches to the query? Similarly, we must prevent the server to return an empty list of results when there are actual matches. This last remark was not taken into account in [SPS14], and the modification the authors present to make their scheme secure against malicious adversaries does not prevent this attack.

### 6.2.1 Verifiable Hash Tables

A central component of our verifiable SSE construction is *verifiable hash tables*. It implements the functionality of a regular (static) hash table, but also provides a proof that, when querying a key in the hash table, the returned element is the right one, and that there are no associated element when querying a key that is not present in the hash table.

More formally, a verifiable hash table is a tuple of algorithms  $\Theta = (\text{VHTSetup}, \text{VHTGet}, \text{VHTVerify}, \text{VHTUpdate}, \text{VHTRefresh})$  :

- $\text{VHTSetup}(T)$  takes as input a hash table  $T$  and outputs  $(K_{\text{VHT}}, \text{VHT}, \sigma_{\text{VHT}})$  where  $K_{\text{VHT}}$  is a private key,  $\text{VHT}$  is the verifiable hash table data structure (possibly including a public key), and  $\sigma_{\text{VHT}}$  the client's state.
- $\text{VHTUpdate}(T, \text{VHT}, u)$  takes as input a hash table  $T$  together with its verifiable data structure  $\text{VHT}$ , and updates all of these according to the update operation  $u$  coming from a predefined update set (e.g. replacement of a value in the table, deletion of a key, ...). It outputs the new verifiable hash table  $\text{VHT}'$  and an update proof  $\pi$ . Its complexity for a table of size  $n$  is denoted  $T_{\text{up}}^{\text{prove}}(n)$ .
- $\text{VHTRefresh}(K_{\text{VHT}}, \sigma_{\text{VHT}}, \pi, u)$  refreshes the digest  $\sigma_{\text{VHT}}$  (i.e. the client's state) according to the update  $u$ , using the proof  $\pi$  generated by a previous  $\text{VHTUpdate}$  call. Its complexity for a table of size  $n$  is denoted  $T_{\text{up}}^{\text{check}}(n)$ .
- $\text{VHTGet}(T, \text{VHT}, \text{hkey})$  outputs the tuple  $(v, \pi)$  where  $v$  is the value associated to  $\text{hkey}$  in  $T$ , and  $\pi$  a proof. For a table of size  $n$ , its complexity is denoted  $T_{\epsilon}^{\text{prove}}(n)$  when  $v \neq \perp$  ( $\text{hkey}$  matches a value), and  $T_{\perp}^{\text{prove}}(n)$  when  $v = \perp$ .
- $\text{VHTVerify}(K_{\text{VHT}}, \sigma_{\text{VHT}}, \text{hkey}, v, \pi)$  returns either ACCEPT or REJECT. For a table of size  $n$ , its complexity is denoted  $T_{\epsilon}^{\text{check}}(n)$  when  $v \neq \perp$ , and  $T_{\perp}^{\text{check}}(n)$  when  $v = \perp$ .

A verifiable hash table must have two properties: correctness ( $\text{VHTGet}$  should return the value associated to  $\text{hkey}$  in  $T$  together with a valid proof) and soundness (it is hard for the server, without  $\sigma_{\text{VHT}}$ , to forge a valid proof, even if he saw a polynomial number of valid proofs, and tried to corrupt the client's state). These properties are formalized by games  $\text{VHTCORR}$  and  $\text{VHTSOUND}$ , as described in Figure 6.2.

The  $\text{VHTCORR}$  is relatively straightforward: it simulates a regular execution of a verifiable hash table (the adversary does not try to cheat by providing adversarial input to the  $\text{VHT}$  functions). The  $\text{VHTSOUND}$  is a bit more involved as it allows for the following behavior: not only the adversary can try to forge a valid proof for an invalid answer (in the Challenge procedure), but he can also try to tamper the client's state by giving him wrong refresh information (in the Update procedure).

**Definition 6.1** (VHT Correctness). *Let  $\Theta$  be a VHT scheme. For an adversary  $A$ , the advantage  $\text{Adv}_{\Sigma, A}^{\text{VHT-corr}}(\lambda)$  of  $A$  in the correctness game is defined as*

$$\text{Adv}_{\Theta, A}^{\text{VHT-corr}}(\lambda) = \mathbb{P}[\text{VHTCORR}_{\Theta}^A(\lambda) = 1].$$

*An VHT scheme  $\Theta$  is correct if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\Theta, A}^{\text{VHT-corr}}(\lambda)$  is negligible in  $\lambda$ .*

<pre> VHTCORR<math>\Theta</math> Init(<math>T</math>)   (<math>K_{\text{VHT}}, \text{VHT}, \sigma_{\text{VHT}}</math>) <math>\leftarrow</math> VHTSetup(<math>T</math>)   <b>return</b> (<math>\text{VHT}, \sigma_{\text{VHT}}</math>) Update(<math>u</math>)   (<math>\text{VHT}', \pi</math>) <math>\leftarrow</math> VHTUpdate(<math>T, \text{VHT}, u</math>)   <math>\sigma'_{\text{VHT}} \leftarrow</math> VHTRefresh(<math>K_{\text{VHT}}, \sigma_{\text{VHT}}, \pi, u</math>)   <b>if</b> <math>\sigma'_{\text{VHT}} \neq \text{REJECT}</math> <b>then</b>     <math>T \leftarrow</math> Apply(<math>T, u</math>)     <math>\text{VHT} \leftarrow \text{VHT}', \sigma_{\text{VHT}} \leftarrow \sigma'_{\text{VHT}}</math>   <b>end if</b>   <b>return</b> (<math>T, \text{VHT}, \sigma_{\text{VHT}}</math>) Challenge(hkey)   (<math>v, \pi</math>) <math>\leftarrow</math> VHTGet(<math>T, \text{VHT}, \text{hkey}</math>)   <b>if</b> VHTVerify(<math>K_{\text{VHT}}, \sigma_{\text{VHT}}, \text{hkey}, v, \pi</math>)     = REJECT or <math>T[\text{hkey}] \neq v</math> <b>then</b>     win <math>\leftarrow</math> true   <b>end if</b>   <b>return</b> (<math>v, \pi</math>) Final()   <b>return</b> win </pre>	<pre> VHTSOUND<math>\Theta</math> Init(<math>T</math>)   (<math>K_{\text{VHT}}, \text{VHT}, \sigma_{\text{VHT}}</math>) <math>\leftarrow</math> VHTSetup(<math>T</math>)   <b>return</b> (<math>\text{VHT}, \sigma_{\text{VHT}}</math>) Query(hkey)   (<math>v, \pi</math>) <math>\leftarrow</math> VHTGet(<math>T, \text{VHT}, \text{hkey}</math>)   <b>return</b> (<math>v, \pi</math>) Update(<math>u</math>)   (<math>\text{VHT}', \pi</math>) <math>\leftarrow</math> VHTUpdate(<math>T, \text{VHT}, u</math>)   The game gives (<math>\text{VHT}', \pi</math>) to the adversary   and gets back <math>\tilde{\pi}</math>   <math>\sigma'_{\text{VHT}} \leftarrow</math> VHTRefresh(<math>K_{\text{VHT}}, \sigma_{\text{VHT}}, \tilde{\pi}, u</math>)   <b>if</b> <math>\sigma'_{\text{VHT}} \neq \text{REJECT}</math> <b>then</b>     <math>T \leftarrow</math> Apply(<math>T, u</math>)     <math>\text{VHT} \leftarrow \text{VHT}', \sigma_{\text{VHT}} \leftarrow \sigma'_{\text{VHT}}</math>   <b>end if</b>   <b>return</b> (<math>T, \text{VHT}, \sigma_{\text{VHT}}</math>) Challenge(hkey, <math>v, \pi</math>)   <b>if</b> VHTVerify(<math>K_{\text{VHT}}, \sigma_{\text{VHT}}, \text{hkey}, v, \pi</math>)     = ACCEPT and <math>T[\text{hkey}] \neq v</math> <b>then</b>     win <math>\leftarrow</math> true   <b>end if</b> Final()   <b>return</b> win </pre>
--	--

**Figure 6.2** – Correctness (VHTCORR – left) and soundness (VHTSOUND – right) games for verifiable hash tables.

**Definition 6.2** (VHT Soundness). *Let  $\Theta$  be an SSE scheme. For an adversary  $A$ , the advantage  $\text{Adv}_{\Theta, A}^{\text{VHT-snd}}(\lambda)$  of  $A$  in the soundness game is defined as*

$$\text{Adv}_{\Theta, A}^{\text{VHT-snd}}(\lambda) = \mathbb{P}[\text{VHTSOUND}_{\Theta}^A(\lambda) = 1].$$

*An SSE scheme  $\Theta$  is sound if for any polynomial-time adversary  $A$ ,  $\text{Adv}_{\Theta, A}^{\text{VHT-snd}}(\lambda)$  is negligible in  $\lambda$ .*

We emphasize that we do not directly need any form of confidentiality on the table content nor on the queries. We will actually start from a secure SSE scheme (in this case security means confidentiality) and turn it into a verifiable SSE scheme. The confidentiality of our scheme will be inherited from the confidentiality of the original scheme. The verifiable hash table that will be put on top of the SSE scheme will ensure soundness. Finally, we have to make sure that correctness of the original scheme correctly transfers to the verifiable scheme.

Construction of verifiable hash table has been extensively studied in the literature, in particular in the case of dynamic tables. For now, we suppose that we have access to an implementation of verifiable hash tables.



### 6.2.2 Static Verifiable Hash Tables

It will also be useful to have *static* verifiable hash tables. In this case, VHTUpdate and VHTRefresh are not implemented, as well as the Update procedures from the security games defined above. Once the hash table is set up, it cannot be modified. These can easily be constructed from a message authentication code.

The idea is to MAC the values with the associated key and the rank of the key in the table (according the lexicographic order of the key). To prove that a key is correctly mapped to a returned value, the server returns the MAC, which the client can easily verify. Because the MAC is secure, the server cannot forge valid tag, and because only a single tag for each position is accessible to the server, he cannot replay them. If the queried key is not present, the server just returns the keys, values and tag of the elements just before and after the expected position of the queried key. He will be able to find this position using binary search.

**Algorithm 6.17** Instantiation of a static verifiable hash table

VHTSetup( $T$ )	VHTVerify( $K, x, v, \pi$ )
<pre> 1: <math>K \leftarrow \mathcal{K}</math> 2: Initialize a empty table VHT of size <math> T </math> 3: <math>i \leftarrow 0</math> 4: <b>for all</b> <math>(key, v) \in T</math> in ascending lexicographic order over <math>key</math> <b>do</b> 5:   <math>s \leftarrow \text{MAC}_K(key, v, i)</math> 6:   <math>\text{VHT}[key] \leftarrow (v, i, s)</math>. 7:   <math>i++</math> 8: <b>end for</b> 9: <b>return</b> <math>(K, \text{VHT})</math> </pre>	<pre> 1: <b>if</b> <math>v \neq \perp</math> <b>then</b> 2:   Parse <math>\pi</math> as <math>(i, s)</math> 3:   <b>if</b> <math>\text{Vf}(K, (x, v, i), s) = \top</math> <b>then</b> 4:     <b>return</b> ACCEPT 5:   <b>else</b> 6:     <b>return</b> REJECT 7:   <b>end if</b> 8: <b>else</b> 9:   Parse <math>\pi</math> as       <math>(i, key_-, v_-, s_-, key_+, v_+, s_+)</math> 10:  <b>if</b> <math>\text{Vf}(K, (key_+, v_+, i + 1), s_+) = \top</math>       <b>and</b> <math>\text{Vf}(K, (key_-, v_-, i), s_-) = \top</math>       <b>and</b> <math>key_- &lt; x &lt; key_+</math> <b>then</b> 11:    <b>return</b> ACCEPT 12:  <b>else</b> 13:    <b>return</b> REJECT 14:  <b>end if</b> 15: <b>end if</b> </pre>
<pre> VHTGet(<math>T, x</math>) 1: <b>if</b> <math>T[x] \neq \perp</math> <b>then</b> 2:   <math>(v, i, s) \leftarrow \text{VHT}[x]</math> 3:   <b>return</b> <math>(v, (i, s))</math> 4: <b>else</b> 5:   Using binary search, locate <math>x</math> in <math>T</math>:       Find <math>i</math> such that <math>key_i &lt; x &lt; key_{i+1}</math>       where <math>(v_i, i, s_i) = \text{VHT}[key_i]</math>       and <math>(v_{i+1}, i + 1, s_{i+1}) = \text{VHT}[key_{i+1}]</math> 6:   <b>return</b> <math>(\perp, (i, key_i, v_i, s_i,</math>       <math>key_{i+1}, v_{i+1}, s_{i+1}))</math> 7: <b>end if</b> </pre>	

The formal description of this construction is given in Algorithm 6.17. Verification will always happen in constant time: there are at most two MACs to verify and two comparisons. The proof generation will have a logarithmic complexity (it will need  $\lceil \log n \rceil$  comparisons when the table contains  $n$  elements) if the key is not present in the table, and constant complexity if it matches a value.

The soundness of the static VHT described in the Algorithm 6.17 directly relies on the security of the MAC, as stated by the following proposition.

**Proposition 6.4.** *Let  $\Theta$  be the static VHT of Algorithm 6.17, and  $A$  a polynomial-time adversary in*

the VHTSOUND game such that the verified table has  $n$  entries and Challenge oracle is called  $q$  times. Then, there is an adversary  $B$  such that

$$\text{Adv}_{\Theta, A}^{\text{VHT-snd}}(\lambda) = \text{Adv}_{\text{MAC}, B}^{\text{euf-cma}}(\lambda)$$

where  $B$  makes  $n$  queries to the Query oracle, and at most  $2 \cdot q$  queries to the Challenge oracle of the EUF-CMA game (cf. Definition 2.8).

*Proof.* Consider the reduction from an adversary  $A$  on the VHT instantiation to the EUF-CMA game adversary  $B$ : every time the instantiation needs to compute  $\text{MAC}_K$ , we replace the call by the Query oracle of the  $G_{\text{euf-cma}}$  game. This only happens during the VHTSetup procedure. When  $A$  produces a forgery  $(key, v, \pi)$ , we will forward it to the  $G_{\text{euf-cma}}$  game and produce a forgery for MAC.

If this forgery is such that  $v \neq \perp$ ,  $\pi$  can be parsed as  $(i, s)$  and hence,  $((key, v, i), s)$  is forwarded to  $G_{\text{euf-cma}}$ . If  $A$  wins the soundness game with this forgery, it implies that

$$(v, \pi) \neq \text{VHTGet}(T, \text{VHT}, key),$$

hence that  $((key, v, i), s)$  is not in the transcript of  $G_{\text{euf-cma}}$  ( $G_{\text{euf-cma}}$ 's Query oracle is called only during the setup phase). It also implies that VHTVerify accepts, *i.e.* that the MAC verification succeeded, and that the forwarded forgery was a valid unseen forgery.

If the forgery is such that  $v = \perp$ ,  $\pi$  can be parsed as  $(i, key_-, v_-, s_-, key_+, v_+, s_+)$ . We know that  $\pi \neq (i^{real}, key_-^{real}, v_-^{real}, s_-^{real}, key_+^{real}, v_+^{real}, s_+^{real})$  where the *real* values are the one generated by  $\text{VHTGet}(T, \text{VHT}, key)$ . We also know that  $key_- < key < key_+$ , because VHTVerify would not accept otherwise. Let us separate two cases:

- $i \neq i^{real}$ : as VHTVerify accepts,  $(key_-, v_-, i, s_-)$  and  $(key_+, v_+, i + 1, s_+)$  cannot be both entries of VHT. If that were the case, it would mean that the condition  $key_- < key < key_+$  is not verified, and the proof  $\pi$  not valid, because, by construction, we know that  $key_-^{real}$  and  $key_+^{real}$  is the only pair of *consecutive* keys verifying this condition, and it would contradict  $i \neq i^{real}$ . Hence, if  $(key_-, v_-, i, s_-)$  is not an entry of VHT, then  $((key_-, v_-, i), s_-)$  does not appear in the transcript of the CMA game and is a valid MAC forgery. The same applies if  $(key_+, v_+, i + 1, s_+)$  is not an entry of VHT.
- $i = i^{real}$ , suppose without loss of generality, that there is a difference on the  $-$  components of the proof. Then  $(key_-, v_-, i, s_-)$  is a valid unseen (because never computed in VHTSetup) MAC forgery.

From the previous arguments, we directly have

$$\text{Adv}_{\Theta, A}^{\text{VHT-snd}}(\lambda) = \text{Adv}_{\text{MAC}, B}^{\text{euf-cma}}(\lambda)$$

□

### 6.2.3 Incremental Hashing

We recall from Section 2.3.5, that a set hashing function is a hash function  $\mathcal{H}$  taking as input sets, with the additional feature that  $\mathcal{H}$  is incremental: it is easy to compute the hash of  $S \cup \{x\}$  or of  $S \setminus \{x\}$  from the hashes of  $S$  from  $x$  (when  $x \notin S$ ). This will be very helpful to efficiently update the SSE verification data structure as the server and/or the client will be able to update the fingerprint of the set of documents matching a keyword very easily, without having to retrieve the entire set and re-hash it.

### 6.3 A Generic and Optimal Construction

In this section, we describe a solution for verifiable symmetric searchable encryption that matches the lower bound of Section 6.1. The construction described later in the section must be seen as a generic solution to the verification problem with SSE: we make use of an SSE instantiation  $\Sigma$  as a black-box and do not rely on its construction. In other words, the data structures built in this section can be used on top (or more exactly besides) of any SSE scheme and turn it in a VSSE scheme with similar confidentiality guarantees and additive logarithmic overhead.

The general principle of this construction is to have a verifiable hash table, whose keys are the database's keywords, and which stores *digests* of an index set. More precisely, we will have a hash table  $T$  such that for any  $w \in W$ ,  $T[w] = \mathcal{H}(\text{DB}(w))$ ,  $\mathcal{H}$  being a hash function defined over sets of strings. The main issue here is to make updates fast: we don't want to re-hash the full set  $\text{DB}(w)$  when updating the database for keyword  $w$ . The hash function  $\mathcal{H}$  must have a certain level of homomorphism: we need a set hashing function.

Let  $\Sigma$  be a dynamic SSE scheme,  $\Theta$  a verifiable hash table instantiation, and  $\mathcal{H}$  a set hashing function. We define  $\text{GVSSSE}_{\Sigma, \Theta, \mathcal{H}}$  (for Generic SSE Verification) as in Algorithm 6.18.

**Correctness.** The correctness of  $\text{GVSSSE}$  is straightforward given the correctness of  $\Sigma$  and completeness of  $\Theta$ . Using a hybrid proof, we can very easily prove the following proposition.

**Proposition 6.5.** *If  $\Sigma$  is a dynamic SSE scheme,  $\Theta$  a verifiable hash table instantiation, and  $\mathcal{H}$  a set hashing function, then for every adversary  $A$ , there exists adversaries  $B$  and  $B'$  such that*

$$\text{Adv}_{\text{GVSSSE}_{\Sigma, \Theta, \mathcal{H}}, A}^{\text{SSE-corr}}(\lambda) \leq \text{Adv}_{\Sigma, B}^{\text{SSE-corr}}(\lambda) + \text{Adv}_{\Theta, B'}^{\text{VHT-corr}}(\lambda)$$

**Soundness.** As expected, the soundness of the  $\text{GVSSSE}$  scheme directly relies on the soundness of  $\Theta$  and the collision resistance of  $\mathcal{H}$ . More formally, we have the following proposition.

**Proposition 6.6.** *If  $\Sigma$  is a dynamic SSE scheme,  $\Theta$  a verifiable hash table instantiation, and  $\mathcal{H}$  a set hashing function, then for every adversary  $A$ , there exists adversaries  $B$ ,  $C$ , and  $D$  such that*

$$\text{Adv}_{\text{GVSSSE}_{\Sigma, \Theta, \mathcal{H}}, A}^{\text{SSE-snd}}(\lambda) \leq \text{Adv}_{\Sigma, B}^{\text{SSE-corr}}(\lambda) + \text{Adv}_{\mathcal{H}, C}^{\text{col}}(\lambda) + \text{Adv}_{\Theta, D}^{\text{VHT-snd}}(\lambda)$$

The idea behind the proof is to use hybrids, in which we successively replace the (sometimes incorrect) SSE scheme by direct calls to the database, the calls to the verifiable hash table by direct call to  $T$  and the set hashing function by a one-to-one representation of the sets.

**Confidentiality.** Confidentiality of the composite scheme that is  $\text{GVSSSE}$  is a little trickier: the information stored in the verifiable hash table, or the keys, give some information to the server even if these were previously hidden by the underlying SSE scheme  $\Sigma$ . For example, the server will immediately learn the number of distinct keywords in the database from the size of the table. Also, he will learn from the search and update queries the repetition of searched/updated keywords (cf. the query pattern in Section 3.2).

**Proposition 6.7.** *Let  $\Sigma$  be a  $\mathcal{L}_{\Sigma}$ -adaptively-secure dynamic SSE scheme,  $\Theta$  a verifiable hash table instantiation, and  $\mathcal{H}$  a set hashing function. Let  $\mathcal{L}_{\text{GVSSSE}}(\text{DB})$  be defined as:*

$$\begin{aligned} \mathcal{L}_{\text{GVSSSE}}^{\text{Stp}}(\text{DB}) &= (\mathcal{L}_{\Sigma}^{\text{Stp}}(\text{DB}), K), \\ \mathcal{L}_{\text{GVSSSE}}^{\text{Srch}}(w) &= (\mathcal{L}_{\Sigma}^{\text{Srch}}(w), \text{qp}(w)), \\ \mathcal{L}_{\text{GVSSSE}}^{\text{Updt}}(\text{op}, \text{in}) &= (\mathcal{L}_{\Sigma}^{\text{Updt}}(\text{op}, \text{in}), \text{op}, \text{qp}(w)). \end{aligned}$$

**Algorithm 6.18** The GVSSE constructionSetup(DB)

- 1:  $(K_\Sigma, \text{EDB}_\Sigma) \leftarrow \Sigma.\text{Setup}(\text{DB})$
- 2: Initialize  $T$  to an empty hash table
- 3:  $K_T, K_S \xleftarrow{\$} \{0, 1\}^\lambda$
- 4: **for all**  $w \in W$  **do**
- 5:    $\text{wtag} \leftarrow F(K_T, w)$
- 6:    $K_e \leftarrow F(K_S, w)$

- 7:    $T[\text{wtag}] \leftarrow \mathcal{H}(F_{K_e}(\text{DB}(w)))$   
     ▷ Apply  $F_{K_e}$  to all the elements of  $\text{DB}(w)$ ,  
     and hash the resulting set
- 8: **end for**
- 9:  $(\text{VHT}, \sigma_{\text{VHT}}) \leftarrow \text{VHTSetup}(T)$
- 10: **return**  $((\text{EDB}_\Sigma, T, \text{VHT}), (K_\Sigma, K_T, K_S), \sigma_{\text{VHT}})$

Search( $K_\Sigma, \sigma, w; \text{EDB}$ )

- 1: Run  $\Sigma.\text{Search}(K_\Sigma, w; \text{EDB}_\Sigma)$ .  
The client gets the result set  $V$
- 2: *Client:*
- 3:  $\text{wtag} \leftarrow F(K_T, w), K_e \leftarrow F(K_S, w)$
- 4: Compute  $\mathcal{H}(F_{K_e}(V))$
- 5: Send  $\text{wtag}$  to the server
- 6: *Server:*
- 7:  $(h, \pi) \leftarrow \text{VHTGet}(T, \text{VHT}, \text{wtag})$
- 8: Send  $(h, \pi)$  to the client
- 9: *Client:*
- 10:  $\text{VHTVerify}(\sigma_{\text{VHT}}, \text{wtag}, h, \pi)$
- 11: **if not**  $h \equiv_{\mathcal{H}} \mathcal{H}(F_{K_e}(V))$
- 12:   **return** REJECT
- 13: **return**  $V$

Update( $K_\Sigma, \sigma, \text{op}, \text{in}; \text{EDB}$ )

- 1: Run  $\Sigma.\text{Update}(K_\Sigma, \text{op}, \text{in}; \text{EDB})$
- 2: For all modified pairs  $(w, \text{ind})$  run the following
- 3: *Client:*
- 4:  $\text{wtag} \leftarrow F(K_T, w), K_e \leftarrow F(K_S, w)$ ,  
 $h' \leftarrow \mathcal{H}(F_{K_e}(\{\text{ind}\}))$
- 5: Send  $(\text{wtag}, h', \text{op})$  to the server
- 6: *Server:*
- 7:  $h \leftarrow T[\text{wtag}]$
- 8: **if**  $\text{op} = \text{add}$  or  $\text{edit}^+, \tilde{h} \leftarrow h +_{\mathcal{H}} h'$
- 9: **if**  $\text{op} = \text{del}$  or  $\text{edit}^-, \tilde{h} \leftarrow h -_{\mathcal{H}} h'$
- 10: Let  $u$  be the replacement of  $h$  by  $\tilde{h}$  in  $T[\text{wtag}]$
- 11:  $(\text{VHT}, \pi) \leftarrow \text{VHTUpdate}(T, \text{VHT}, u)$
- 12: Send  $(\pi, h, \tilde{h})$  to the client
- 13: *Client:*
- 14:  $\sigma_{\text{VHT}} \leftarrow \text{VHTRefresh}(\sigma_{\text{VHT}}, u)$
- 15: **return**  $\sigma_{\text{VHT}}$

Then for every adversary  $A$ , there exist two adversaries  $B$ , and  $C$ , and simulators  $S$  and  $S_\Sigma$  such that

$$\text{Adv}_{\text{GVSSE}_{\Sigma, \Theta, F, \mathcal{H}, S, \mathcal{L}_{\text{GVSSE}}, A}}^{\text{SSE-sim}}(\lambda) \leq \text{Adv}_{\Sigma, S_\Sigma, \mathcal{L}_\Sigma, B}^{\text{SSE-sim}}(\lambda) + \text{Adv}_{F, C}^{\text{prf}}(\lambda)$$

where  $B$  makes the same queries to  $\Sigma$  as the ones  $A$  makes to GVSSE, and where  $C$  makes at most  $K$  queries to  $F$ .

The confidentiality of the  $\Sigma$  is considered under the definition of Section 3.1.3.3, i.e. against an active adversary, which is formally different from the previous confidentiality definitions which considered only passive adversaries. However, we can see that, when  $\Sigma$  has single round Search and Update protocols, these are equivalent (the adversary cannot mess with the first and single message the client sends).

It is also crucial to see that GVSSE cannot be forward private, independently of the chosen SSE scheme: the update part of the leakage function always return the query pattern.

**Computational complexity.** To evaluate the computational complexity of the construction, we will use the dynamic SSE construction  $\Pi_{\text{bas}}^{\text{dyn}}$  of Cash *et al.* [CJJ+14] which has search complexity linear in the number of results and constant update time.

The computational complexity of Search and Update queries in GVSSE is given in Table 6.1 for two possible VHT instantiations, hash-based [TT05] or cryptographic accumulators-based [PTT09]. In the first case, we achieve logarithmic complexity in both accessing and updating (including the verification), and reach the lower bound of Section 6.1.3. The second case achieves optimal search time (for both the client and the server), linear in the number of results, as both access and verification of the VHT is done in constant time, but needs  $\mathcal{O}(K^\varepsilon)$  update time, where  $0 < \varepsilon < 1$  is a fixed constant (for the server only, the client having constant update time). The we could also use the VHT instantiation of [PTT09] which reverses the complexity for searches and update, and end up with an optimal update complexity.

**Table 6.1** – Computational complexity of GVSSE with  $\Pi_{\text{bas}}^{\text{dyn}}$  as SSE scheme, in function of the VHT instantiation used. The update complexities are given for a single modified keyword/document pair. Remind that  $K$  is the number of distinct keywords,  $0 < \varepsilon < 1$  is a fixed constant.

VHT Instantiation	Search		Update	
	Server	Client	Server	Client
Hash Tree [TT05]	$\mathcal{O}(n_w + \log K)$	$\mathcal{O}(n_w + \log K)$	$\mathcal{O}(\log K)$	$\mathcal{O}(\log K)$
Accumulators [PTT09]	Pairing	$\mathcal{O}(n_w + 1/\varepsilon)$	$\mathcal{O}(K^\varepsilon)$	$\mathcal{O}(1/\varepsilon)$
	RSA	$\mathcal{O}(n_w + K^\varepsilon)$	$\mathcal{O}(n_w + 1/\varepsilon)$	$\mathcal{O}(1/\varepsilon)$

Hence, depending on VHT implementations, the GVSSE construction achieves optimality in two senses: first in a general way, showing that the lower bound of Section 6.1 is tight, then for the Search query only, and finally for the Update query only, showing that we can have a VSSE scheme with no asymptotic verification overhead for either search or update queries. We emphasize that the accumulator-based instantiation might be interesting in practice for very large databases with a few updates.

We built a proof of concept to evaluate the performance of GVSSE, using a hash-tree-based VHT, and ECMH [MTA16] for the set hash function. The verification takes about 3 ms for a query matching 1000 results, and 708 ms for 200 000 results, on a database with  $4.6 \times 10^6$  distinct keywords. The set hashing function hashes an element in less than 3.6  $\mu\text{s}$ . Our experiments were using a single thread and were running on a Xeon at 2.66 GHz.

## 6.4 Verifying $\Sigma\phi\phi\phi$ , Diana, and Janus

In Chapters 4 and 5, we showed the security of  $\Sigma\phi\phi\phi$ , Diana and Janus, but only against passive adversaries. However, all these schemes can easily be turned into verifiable SSE schemes.

We can do so by storing a hash of  $\text{DB}(w)$  for every  $w$  on the client, and during a search, the stored hash value is checked to match the hash recomputed from the server's results. Again, this hash will be computed with a set hashing function to allow for easy incremental updates.

Because the hashes are locally stored, only the client will be impacted by this modification, both on the computational complexity, and on storage complexity of the schemes. Yet, as we purposely

used an incremental hash function, the update complexity will not be impacted, neither will be the search complexity, as it is always (at least) linear in the number of updates. The client will have to store additional  $\mathcal{O}(\lambda)$  bits per keyword (the hash value).

As for the original schemes, we could outsource this storage using ORAM. Yet, while in Section 4.5.7, we used regular ORAM, here we need to use verifiable ORAM [AKST14] to ensure that the server always answers correctly to the queries made to the ORAM in which the hashes are stored.

## 6.5 Verifying SPS

In this section, we show how to make the SPS scheme, authored by Stefanov *et al.* [SPS14] verifiable. Note that this section is quite technical, and is not absolutely necessary for the understanding of this thesis.

SPS relies on a hierarchical structure to ensure forward privacy. As explained in Section 4.4, we can see SPS as levels of static SSE which are rebuilt upon modification of the database. Hence, we do not really need some dynamic verifiable data structures, static ones will suffice.

### 6.5.1 Remembering SPS

The SPS construction [SPS14] consists of  $L + 1$  levels  $\mathbf{T}_0, \dots, \mathbf{T}_L$ , where  $L = \lfloor \log N \rfloor$ , and each level  $\mathbf{T}_\ell$  is of size  $2^\ell$ . Each level can be seen as a map between keywords and lists of tuples of the form  $(\text{ind}, \text{op}, \text{cnt})$ , where  $\text{ind}$  is a document index,  $\text{op} \in \{\text{add}, \text{del}\}$  the operation associated with the tuple (addition or deletion) and  $\text{cnt}$  a unique counter. These tuples are encoded in a way that their content is hidden unless given a token allowing decryption. This token will be unique per keyword  $w$  and per level  $\ell$ , and with such a token, the server will not learn anything else than the content of the tuples associated with the keyword  $w$  at level  $\ell$ . They are actually stored in a hash table, that allows for a constant-time lookup, for a specific tuple  $(w, \text{op}, \text{cnt})$ , providing either the document  $\text{ind}$  or  $\perp$  as output. This conceptual lookup operation is denoted  $\Gamma_\ell[w, \text{op}, \text{cnt}]$ . Moreover, in a level  $\ell$ , all the tuples  $(w, \text{ind}, \text{op}, \text{cnt})$  are lexicographically sorted based on key  $(w, \text{ind}, \text{op})$ .

For dynamism, the update operation exploits the leveled structure: for a new entry (addition or deletion), it is either inserted at level 0, in the unique cell of  $\mathbf{T}_0$ , or in the first empty level  $\mathbf{T}_\ell$  with all the entries from the lower levels  $\mathbf{T}_0, \dots, \mathbf{T}_{\ell-1}$ : the  $2^\ell - 1$  entries in the lower levels and the new entry are indeed merged and re-ordered in  $\mathbf{T}_\ell$  in the lexicographic ordering based on  $(w, \text{ind}, \text{op})$ , using an oblivious sort [GM11]. Once the tuples are sorted, one can easily cancel addition-deletion for the same document/keyword pair, just replacing the two entries by  $\perp$ . The previous levels are all emptied.

To perform a search with keyword  $w$ , for each level, the server is provided the token for  $w$ , and performs lookups to find all matching entry in the levels. He does that for both add and del operations, simplifying inserted-then-deleted document/keyword pairs from the result set. However, such a data structure can lead to a linear-time search, in the worse case.

They thus improved their approach with an additional information in the tuples. The main drawback with the above tuples is the del operations that have to be looked for in a different level before returning a document that has been added and might have been deleted later. Also an extra information is added to del entries, such that, at level  $\ell$ , given an add tuple with no matching del entry, the server can efficiently find the next add tuple with no matching del entry. This extra information is the field  $\ell^*$ , called the *target level*, and the conceptual lookup operation now return  $\Gamma_\ell[w, \text{op}, \text{cnt}] = (\ell^*, \text{ind})$  or  $\perp$ . In a deletion tuple,  $\ell^*$  is the level where the associated addition tuple is stored. In an addition tuple, we just set  $\ell^* = \ell$ . If each level is now lexicographic ordered based

on  $(w, \ell^*, \text{ind}, \text{op})$ , a fast procedure allows the server to find an addition tuple without deletion, leading to a complexity in  $\mathcal{O}(m \log^3 N)$ . It can find and skip holes (a series of addition tuples that have been deleted later) in each level. Unfortunately, as is, these methods do not provide verifiable outputs to the client. Also note that SPS only supports add and del update operations, not  $\text{edit}^+$  nor  $\text{edit}^-$ .

Stefanov *et al.* briefly described how to make their construction secure in the malicious model. However, although the modifications they propose are essential in the malicious model, these are not sufficient to guarantee security against active adversaries.

Also, note that the generic framework that we developed above cannot be applied to SPS without loosing the forward privacy property. Namely, repeated accesses to the (authenticated) dictionary storing the hash of the result set leak: authenticated dictionaries are used to ensure the integrity of the stored data, but not to hide the access pattern. Because we need to update the hash of the result set  $\text{DB}(w)$  when updating keyword  $w$ , searching for  $w$ , and then updating  $w$  would leak that the updated keyword has been previously searched, break the forward privacy of SPS.

We could use some kind of verifiable ORAM (*i.e.* ORAM secure against malicious servers), but that would add a non negligible overhead. Instead, we will see how to fix the original SPS proposal.

### 6.5.2 Quick Cryptanalysis of SPS

In [SPS14], the authors claim that adding a timestamp to the entries (as we did), MAC-ing these, returning to the client the MACs with each non deleted entry and the holes proofs (*i.e.* the del entries at the edge of each hole component, together with their MAC) is enough. The client then would have to check the size of the holes and check that they match the distance between two add entries (as CheckResults does).

**Incompletely scanned levels.** One key modification brought by our protocol Verif-SPS is the verification that the search has been performed over all the entries of a specific level. The absence of such proofs (which is the role of non-member proofs in Verif-SPS) allows for trivial attacks where the server omits to return part of the results (*e.g.*, the last add entry of a level), resulting in missing results, or part of the holes (*e.g.* the last hole component of a level), resulting in non matching results.

We will see that the generation of such proofs in Verif-SPS is the main modification compared to the original SPS scheme, and that this modification implies the logarithmic overhead of the basic verifiable scheme.

**Non-consecutive holes.** Suppose now that the previous issue is (somehow) fixed. Unfortunately, there is still a possible attack in the claimed security against active adversaries of SPS.

In fact, checking the size of the holes is not enough to protect against malicious servers. Say there is, at level  $\ell$ , an add entry  $e_*^+$  in a hole between add entries  $e_1^+$  and  $e_2^+$ , and that matches a del entry  $e_*^-$ . For the sake of simplicity, we suppose that the del entries matching the hole between  $e_1^+$  and  $e_2^+$  are all at the same level, between entries  $e_1^-$  and  $e_2^-$ . The server could decide to split the real hole in two, and make the client believe that  $e_*^+$  is a non deleted matching add entry. He will, in some sort, omit the existence of  $e_*^-$ .

However, the size of the hole will still be equal to the distance between the returned add entries ( $e_1^+$  and  $e_*^+$  for the first hole, and  $e_*^+$  and  $e_2^+$  for the second one). In the sublinear variant of Verif-SPS, the attack is thwarted by a procedure CheckAdjacency. It verifies that every hole component is

either the last of its level, or immediately followed by another hole component, and hence that no del entry has been omitted by the server.

### 6.5.3 Verifiable SPS: Basic Construction

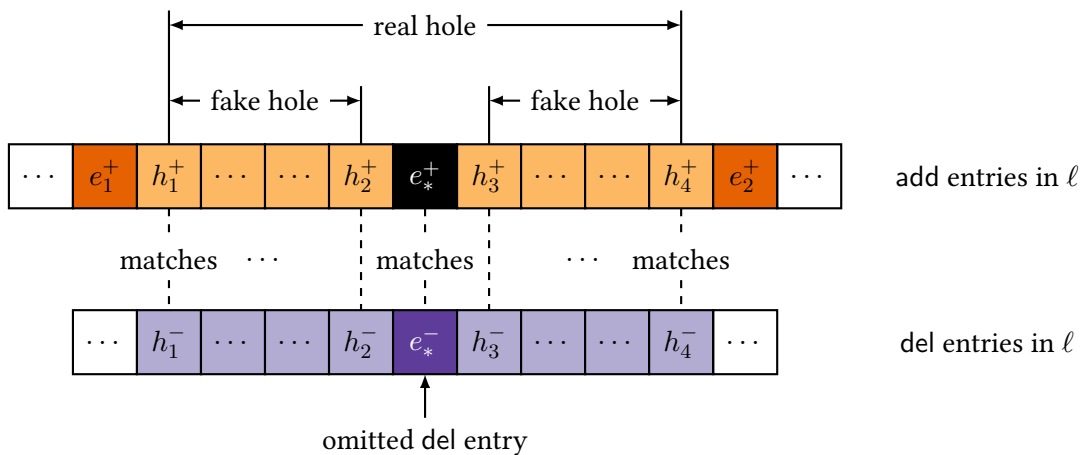
To avoid the server tampering with the data, one could MAC the tuples in the levels. This would ensure that the server never returns to the client a tuple that was not in the level. Yet, it would not prevent the server from not returning a matching tuple (either for addition or deletion) for a search query.

Using verifiable hash tables and the particularities of the SPS construction, we show how we can thwart this attack, and more generally how to make SPS verifiable at the extra cost of a verifiable hash table per level.

To give a better insight into our verifiable SSE instantiation, we first give a basic construction, itself based on the basic construction of [SPS14]. The modified construction is described in Algorithms 6.19, 6.20, and 6.21. It uses a static verifiable hash table instantiation  $\Theta = (\text{VHTSetup}, \text{VHTGet}, \text{VHTVerify})$ . Except for Search, the modifications are highlighted in underlined red.

For all of the algorithms/protocols, except Search, the main modification resides in the addition of a verifiable hash table at every level and the use of authenticated encryption. For example, the Lookup function, in addition to retrieving the entry associated with the search token token, operation op, and counter cnt, will also return a proof that the returned value is correct.

The modifications brought to the Rebuild algorithm (Algorithm 6.20) are only a bit more complicated to analyze. First, the use of authenticated encryption instead of encryption without integrity verification ensures that the server only sends valid entries to the client. However, it does not prevent the server from sending the entries in the wrong order, or to send the same entry twice. To avoid such attacks, we need to ensure that the oblivious sort algorithm is secure, even in the presence of a malicious server. But we also have to make sure that the server cannot replay ciphertexts generated in the past to fool the client. So, we add to the additional data a per-level timestamp  $r_\ell$ , incremented every time a level is rebuilt, as well as a flag, set to 0 or 1, to distinguish the ciphertexts produced before and after the first call to o-sort. Finally, the authenticating data structure for the verifiable hash table is initialized (line 26).



**Figure 6.3** – Illustration of a non-consecutive hole attack. The adversary returns  $e_*^+$  as a (fake) additional result, together with the proofs for non-maximal holes  $(h_1^-, h_2^-)$  and  $(h_3^-, h_4^-)$ .



---

**Algorithm 6.19** The Verif-SPS construction, derived from SPS [SPS14].
 

---

**Setup(DB)**

- 1: Client chooses an encryption key  $esk$ ,  $L + 1$  random level keys  $k_0, \dots, k_L$  (where  $L = \log N$ ), and a key  $K_e$  for the PRF  $F$ . The scheme's key is  $K_\Sigma = (esk, k_0, \dots, k_L, K_e)$ .
- 2: Client initializes an empty hierarchical structure  $D$  consisting of exponentially growing levels  $\mathbf{T}_0, \dots, \mathbf{T}_L$ .
- 3: **Initialize counters**  $r_0, \dots, r_L$  to 0.
- 4: **return**  $(K_\Sigma, (r_0, \dots, r_L), (\mathbf{T}_0, \dots, \mathbf{T}_L))$

**Lookup(token, op, cnt)**

- 1:  $hkey \leftarrow H_{\text{token}}(0||\text{op}||\text{cnt})$
- 2:  $(v, \pi) \leftarrow \text{VHTGet}(\mathbf{T}_\ell, \text{VHT}_\ell, hkey)$
- 3: **if**  $v = \perp$  **then return**  $(\perp, \pi)$
- 4: **else**
- 5:   Parse  $v$  as  $(c_1, c_2)$ .
- 6:    $(\ell^*, \widetilde{\text{ind}}) \leftarrow c_1 \oplus H_{\text{token}}(1||\text{op}||\text{cnt})$ .
- 7:   **return**  $(\ell^*, \widetilde{\text{ind}}, v, \pi)$
- 8: **end if**

**EncodeEntry $_{esk, \ell}(w, \ell^*, \text{ind}, \text{op}, \text{cnt})$** 

- 1:  $\text{token}_\ell \leftarrow F(k_\ell, h(w))$
- 2:  $hkey \leftarrow H_{\text{token}_\ell}(0||\text{op}||\text{cnt})$
- 3:  $c_1 \leftarrow (\ell^*, \text{ind}) \oplus H_{\text{token}_\ell}(1||\text{op}||\text{cnt})$
- 4:  $c_2 \leftarrow \text{AEnc}(esk, (\ell, r_\ell),$
- 5:          $(w, \ell^*, \text{ind}, \text{op}, \text{cnt}))$
- 6: **return**  $(hkey, c_1, c_2)$

**Update( $K_\Sigma$ , op, ind, W; EDB)**

- 1: **for all**  $w \in W$  in random order **do**
- 2:   Compute the target level  $\ell^*$  of  $(w, \text{ind}, \text{op})$
- 3:   **if**  $\mathbf{T}_0$  is empty **then**
- 4:     Select a fresh key  $k_0$ .  $r_0++$ .
- 5:      $\mathbf{T}_0 \leftarrow \text{EncodeEntry}_{esk, 0}(w, \ell^*, \text{ind}, \text{op})$
- 6:   **else**
- 7:     Let  $\mathbf{T}_\ell$  denote the first empty level.
- 8:      $r_\ell++$ . Rebuild $(\ell, (w, \text{ind}, \text{op}))$ .
- 9:   **end if**
- 10: **end for**

**Oblivious sorting in presence of a malicious server.** Usually, the problem of oblivious sorting is considered in the semi-honest setting, where the server only tries to infer information from the messages he sees, but sticks to the protocol's execution.

A solution to enable resistance against active adversaries would be to rely on memory checking techniques, but as we saw in Section 6.1.2, these are costly and would incur a logarithmic overhead. We could use less general verification techniques using the fact that, in the case of oblivious sorting, the client always knows which memory locations were modified at what time, as I/Os are independent from the sorted values. The client could use this property and add metadata – a timestamp and the memory location – to each memory block used by the algorithm. Every time a block is processed, the client will check that these data are consistent with the algorithm state. For example, when using sorting networks [GM11], we can MAC the location in the array and an integer accounting the last comparison executed on the item (in a sorting network, we can assign to each comparison a unique integer) to the probabilistic encryption of the item (used to ensure the obliviousness of the protocol). Every time an item is used by the algorithm, the client checks this MAC and ensures that it is not an element forged by the adversary. Using authenticated encryption with additional data, we could actually do both MAC and encryption at once, and the additional cost (compared to pure encryption) would be negligible using modern AEAD schemes (e.g. [RBB03]). For now, we suppose that o-sort is secure against active adversaries.

---

**Algorithm 6.20** Modified Rebuild protocol of the Verif-SPS construction.

---

Rebuild( $\ell, (w, \ell^*, \text{ind}, \text{op}, \text{cnt})$ ).

```

1: Let  $\text{entry}^* \leftarrow \text{EncodeEntry}_{esk, K_e, k_0}(w, \ell^*, \text{ind}, \text{op}, \text{cnt})$ 
2: Let  $\hat{\mathbf{B}} \leftarrow \{\text{entry}^*\} \cup \mathbf{T}_0 \cup \dots \cup \mathbf{T}_{\ell-1}$ .
3: for all  $\text{entry} = (\text{hkey}, c_1, c_2) \in \hat{\mathbf{B}}$ , with original level  $\ell'$  do
4:    $(w, \ell^*, \text{ind}, \text{op}, \text{cnt}) \leftarrow \text{ADec}(esk, (\ell', r_{\ell'}), c_2)$ 
5:   Overwrite entry with  $\text{AEnc}(esk, (\ell, r_\ell, 0), (w, \ell^*, \text{ind}, \text{op}, \text{cnt}))$ 
6: end for
7:  $\hat{\mathbf{B}} \leftarrow \text{o-sort}(\hat{\mathbf{B}})$ , based on the lexicographic sorting key  $(w, \ell^*, \text{ind}, \text{op})$ .
8: for all  $e = \text{AEnc}(esk, (\ell, r_\ell, 0), (w, \ell^*, \text{ind}, \text{op}, \text{cnt}))$  in  $\hat{\mathbf{B}}$  in sorted order do
9:   if  $e$  marks the start of a new word  $w$ , for an operation  $\text{op} \in \{\text{add}, \text{del}\}$  then
10:    Set  $\text{cnt}_{\text{op}, w} \leftarrow 0$ 
11:     $e \leftarrow \text{AEnc}(esk, (\ell, r_\ell, 1), (w, \ell^*, \text{ind}, \text{op}, 0))$ 
12:   else if  $e$  and its adjacent entry are add and del operations for the same  $(w, \ell^*, \text{ind})$  pair then
13:    Suppress the entries by updating both entries with  $\text{AEnc}(esk, (\ell, r_\ell, 1), \perp)$ 
14:   else Update  $e$  in  $\hat{\mathbf{B}}$ :
15:     $e \leftarrow \text{AEnc}(esk, (\ell, r_\ell, 1), (w, \ell^*, \text{ind}, \text{op}, \text{cnt}_{\text{op}, w}++))$ 
16:   end if
17: end for
18: Randomly permute  $\hat{\mathbf{B}} \leftarrow \text{o-sort}(\hat{\mathbf{B}})$ , based on  $\text{hkey}$ .
19: Select a new level key  $k_\ell$ .
20: for all  $\text{entry} \in \hat{\mathbf{B}}$  do
21:    $(w, \ell^*, \text{ind}, \text{op}, \text{cnt}) \leftarrow \text{ADec}(esk, (\ell, r_\ell, 1), \text{entry})$ 
22:   if  $\text{op} = \text{add}$ ,  $\ell^* \leftarrow \ell$ 
23:    $(\text{hkey}, c_1, c_2) \leftarrow \text{EncodeEntry}_{esk, K_e, k_\ell}(w, \ell^*, \text{ind}, \text{op}, \text{cnt})$ 
24:   Add  $(\text{hkey}, c_1, c_2)$  to  $\mathbf{T}_\ell$ 
25: end for
26:  $(\text{VHT}_\ell, \sigma_{\text{VHT}_\ell}) \leftarrow \text{VHTSetup}(\mathbf{T}_\ell)$ 

```

---

**Verifiable search.** The idea in the simple search algorithm is to make the server send to the client *all* the entries matching the searched keyword at every level  $\ell$ , both for  $\text{op} = \text{add}$  and  $\text{op} = \text{del}$ , make the client perform the eliminations among these entries.

To verify that the server sent all the entries corresponding to keyword  $w$  with operation  $\text{op}$  at level  $\ell$ , we use the fact that the counters used to compute the entries'  $\text{hkey}$  are consecutive. So, if there are  $c$  add entries at level  $\ell$ , for all  $0 \leq i < c$ ,  $\Gamma_\ell[w, \text{add}, i] \neq \perp$ , and  $\Gamma_\ell[w, \text{add}, c] = \perp$ . Consecutiveness of the  $\text{cnt}_{\text{op}, w}$  values, is ensured by the soundness of the o-sort algorithm.

Finally, as we check that the entries have the right value (and in particular that the last one is  $\perp$ ), if the protocol does not abort, we are sure that  $\tilde{\mathcal{I}}_\ell^+$  contains all the  $(w, \text{add})$  entries at level  $\ell$ , and that  $\tilde{\mathcal{I}}_\ell^-$  contains all the  $(w, \text{del})$  entries at level  $\ell$ .

### 6.5.4 Sublinear Construction

We can also fix the sublinear version of SPS, in order to thwart the attacks presented in Section 6.5.2. In particular, we modify the server so that he can prove to the client that he completely scanned all levels, and that the returned holes are indeed consistent with the returned results.

---

**Algorithm 6.21** Simple Search algorithm for Verif-SPS. Every time that the result of a client's computation is REJECT, he immediately stops and returns REJECT.

---

Search( $K_\Sigma, w, \sigma; \text{EDB}$ )

---

```

1: Client:
2:  $\text{tk} \leftarrow \{\text{token}_\ell = F(k_\ell, h(w)), \ell = 0, \dots, L\}$   $\triangleright$  Compute tokens for each level
3: The client sends  $\text{tk}$  to the server.

   Server:
4: for  $\ell = L$  to  $0$  do
5:   Initialize  $\tilde{\mathcal{I}}_\ell^+$  and  $\tilde{\mathcal{I}}_\ell^-$  to empty lists.  $\text{cnt} \leftarrow 0$ 
6:   repeat
7:      $(\ell^*, \text{ind}, \text{entry}, \pi) \leftarrow \text{Lookup}(\text{token}_\ell, \text{add}, \text{cnt}++)$ ,  $\tilde{\mathcal{I}}_\ell^+[\text{cnt}] \leftarrow (\text{entry}, \pi)$ 
8:   until  $\text{entry} = \perp$ 
9:    $\text{cnt} \leftarrow 0$ 
10:  repeat
11:     $(\ell^*, \text{ind}, \text{entry}, \pi) \leftarrow \text{Lookup}(\text{token}_\ell, \text{del}, \text{cnt}++)$ ,  $\tilde{\mathcal{I}}_\ell^-[\text{cnt}] \leftarrow (\text{entry}, \pi)$ 
12:  until  $\text{entry} = \perp$ 
13:  Send  $\{(\tilde{\mathcal{I}}_\ell^+, \tilde{\mathcal{I}}_\ell^-)\}_\ell$  to the client.
14: end for

   Client:
15: Let  $\mathcal{I} \leftarrow \emptyset$ 
16: for  $\ell = L$  to  $0$  do
17:   for  $i = 0$  to  $|\tilde{\mathcal{I}}_\ell^+| - 2$  do  $\triangleright$  Check that all the elements (but the last) of the list are valid elements
18:     Parse  $\tilde{\mathcal{I}}_\ell^+[i]$  as  $(\text{entry}, \pi)$ 
19:      $\text{hkey} \leftarrow H_{\text{token}_\ell}(0||\text{add}||i)$ 
20:     if  $\text{VHTVerify}(\sigma_{\text{VHT}_\ell}, \text{hkey}, \text{entry}, \pi) = \text{REJECT}$  or  $\text{entry} = \perp$  then
21:       return REJECT
22:     else
23:        $(\ell^*, \text{ind}) \leftarrow \text{entry}.c_1 \oplus H_{\text{token}_\ell}(1||\text{add}||i)$ ,  $\mathcal{I} \leftarrow \mathcal{I} \cup \{\text{ind}\}$ 
24:     end if
25:   end for
26:   Parse  $\tilde{\mathcal{I}}_\ell^+[(|\tilde{\mathcal{I}}_\ell^+| - 1)]$  as  $(\text{entry}, \pi)$   $\triangleright$  Check that the last element of the list is  $\perp$ 
27:    $\text{hkey} \leftarrow H_{\text{token}_\ell}(0||\text{add}||(|\tilde{\mathcal{I}}_\ell^+| - 1))$ 
28:    $\text{VHTVerify}(\sigma_{\text{VHT}_\ell}, \text{hkey}, \perp, \pi)$ 
29:   Do the same as before with del instead of add, running the loop over  $\tilde{\mathcal{I}}_\ell^-$  instead of  $\tilde{\mathcal{I}}_\ell^+$ , and removing elements to  $\mathcal{I}$  instead of adding them.
30: end for
31: return  $\mathcal{I}$ 

```

---

As in the simple search algorithm, the main threat the client has to protect against is not receiving all the elements (either the add entries or the holes) necessary to compute the search result. Again, we heavily rely on the structure of the levels and on how the entries' counter increases. For example, for each level  $\ell$ , the server will give a proof that for a counter  $\text{cnt}_{\text{max}}$ , the entry  $\Gamma_\ell[w, \text{add}, \text{cnt}_{\text{max}}]$

is  $\perp$ , ensuring that there is no add entry matching  $w$  with counter  $cnt > cnt_{\max}$ . We proceed similarly for del entries.

The verification of the sublinear Search algorithm is not as direct as the verification of basic Search, and we will present all the procedures needed for the generation of proofs and their verification. For the sake of readability, in the algorithms, as before, we suppose that once a call returned REJECT, the calling procedure immediately stops and returns REJECT itself. We also omit the parameters  $K, w, \sigma$  and EDB in the procedure signatures.

The Search algorithm, as described by Algorithm 6.22, calls four sub-procedures:

- ProcessLevel, as in the original SPS scheme, finds all the tuples  $(w, \ell, \text{ind}, \text{add})$  stored at level  $\ell$  such that there is no corresponding entry  $(w, \ell, \text{ind}, \text{del})$  in lower levels  $\ell' < \ell$ . To do so, it has to find the holes with target level  $\ell$ , so, for verification purposes, it also returns these holes and adds them to the hash table AllHoles that tracks all the holes ever found.
- ProveHoles takes as input all the holes found by the calls to ProcessLevel and returns a proof for these, *i.e.* a proof that they are *maximal* and *consecutive* (we will explain these notions later).
- VerifyHoles verifies the proofs produced by ProveHoles and returns the corresponding holes.
- CheckResults checks the consistency of the results returned by ProcessLevel and produces the result set  $\mathcal{I}$ .

---

**Algorithm 6.22** Sublinear Search algorithm.

---

Search( $K_\Sigma, w, \sigma$ ; EDB)

---

- 1: *Client*: On input  $(K_\Sigma, w)$ , it computes tokens for each level.
  - 2:  $\text{tk} \leftarrow \{\text{token}_\ell = F(k_\ell, h(w)), \ell = 0, \dots, L\}$
  - 3: The client sends  $\text{tk}$  to the server.
  - 4: *Server*:
  - 5: Initialize AllHoles as an empty hash table of lists.
  - 6: **for**  $\ell = 0$  **to**  $L$  **do**
  - 7:    $\mathcal{I}_\ell \leftarrow \text{ProcessLevel}(\ell, \text{token}_\ell)$
  - 8: **end for**
  - 9:  $\Pi_{\text{Holes}} \leftarrow \text{ProveHoles}(\text{AllHoles})$
  - 10: Send  $(\mathcal{I}_0, \dots, \mathcal{I}_L, \Pi_{\text{Holes}})$  to the client.
  - 11: *Client*:
  - 12:  $\text{AllHoles} \leftarrow \text{VerifyHoles}(\Pi_{\text{Holes}})$
  - 13:  $\mathcal{I} \leftarrow \text{CheckResults}(\mathcal{I}_0, \dots, \mathcal{I}_L, \text{AllHoles})$
  - 14: **return**  $\mathcal{I}$
- 

**The ProcessLevel algorithm.** ProcessLevel is the key procedure described in Algorithm 6.23. It goes through the add entries for keyword  $w$  at level  $\ell$  and looks for those that were not deleted by a subsequent deletion. If such a deletion happened, a corresponding  $(w, \ell, \text{ind}, \text{del})$  entry will be in some lower level  $\ell' < \ell$  (note that such an entry cannot be stored at level  $\ell$  because it would have been “simplified” by the Rebuild algorithm). If this search for deletions were done on an

entry-by-entry basis, it would be very inefficient, and the server would need to go through all the add entries at level  $\ell$  and the search complexity would not be better than in the simple Search protocol.

To avoid this, if ProcessLevel found an entry that was deleted in the lower levels, it will directly jump to the next add entry not deleted. This is where SkipHole enters: it finds the largest collection of successive add entries at level  $\ell$  (starting at the current entry) for which we can find a matching collection of del entries in lower levels (line 1). These entries are called a *hole*.

---

**Algorithm 6.23** The ProcessLevel algorithm (and auxiliary procedures).

---

ProcessLevel( $\ell$ , token $_\ell$ )

```

1:  $cnt \leftarrow 0$ 
2: Initialize  $\mathcal{I}_\ell$  as an empty list.
3:  $(\ell, \text{ind}, \text{entry}, \pi) \leftarrow \text{Lookup}(\text{token}_\ell, \text{add}, cnt++)$ 
4: while  $\text{entry} \neq \perp$  do
5:   if  $(w, \ell, \text{ind}, \text{del})$  is not found in any lower levels then  $\triangleright$  Through a binary search for each lower level
6:     Append  $(\text{entry}, \pi)$  to  $\mathcal{I}_\ell$ 
7:   else
8:     Call  $(cnt, \text{Hole}) \leftarrow \text{SkipHole}(\ell, \text{token}_\ell, \text{ind})$ 
9:     Append Hole to  $\mathcal{I}_\ell$ 
10:  end if
11:   $(\ell, \text{ind}, \text{entry}, \pi) \leftarrow \text{Lookup}(\text{token}_\ell, \text{add}, cnt++)$ 
12: end while
13: Append  $(\text{entry} = \perp, \pi)$  to  $\mathcal{I}_\ell$   $\triangleright$  Show that we reached the last add element in level  $\ell$ .
14: return  $\mathcal{I}_\ell$ 

```

AppendHole(AllHoles, Hole)

```

1: for all  $(\ell', \ell, cnt_x, cnt_y, \text{ind}_x, \text{ind}_y) \in \text{Hole}$  do
2:   Append  $(\ell, cnt_x, cnt_y)$  to AllHoles[ $\ell'$ ]
3: end for

```

SkipHole( $\ell$ , token $_\ell$ , ind)

```

1: Through binary search, compute the maximum identifier  $\text{ind}' \geq \text{ind}$  in level  $\ell$  s.t.
    $\text{count}_{\ell, \ell, w, \text{add}}(\text{ind}', \text{ind}) + 1 = \text{sum}$ , with  $(\text{sum}, \text{Hole}) \leftarrow \text{DelSum}(\ell, \text{ind}, \text{ind}')$ 
2: AppendHole(AllHoles, Hole)  $\triangleright$  Track all the generated holes.
3: return the corresponding value  $cnt$  for  $\text{ind}'$  and Hole.

```

DeletedSum( $\ell$ , ind, ind')

```

1:  $\text{sum} \leftarrow 0$ , Hole  $\leftarrow$  empty list
2: for  $\ell' = 0$  to  $\ell - 1$  do
3:   Find the largest region  $[(\ell, \text{ind}_x), (\ell, \text{ind}_y)]$  that falls within the range  $[(\ell, \text{ind}), (\ell, \text{ind}')]$ 
4:   if such a region is found then
5:     Let  $cnt_x$  and  $cnt_y$  be such that  $(\ell, \text{ind}_x) = \Gamma_\ell[w, \text{del}, cnt_x]$  and  $(\ell, \text{ind}_y) = \Gamma_\ell[w, \text{del}, cnt_y]$ 
6:      $\text{sum} \leftarrow \text{sum} + cnt_y - cnt_x + 1$ 
7:     Append  $(\ell', \ell, cnt_x, cnt_y, \text{ind}_x, \text{ind}_y)$  to Hole
8:   end if
9: end for
10: return  $(\text{sum}, \text{Hole})$ .

```

---

For this, it uses DeletedSum. DeletedSum finds in every level  $\ell' < \ell$  the largest region of successive del entries whose target level is  $\ell$  and document's index comprised between  $\text{ind}$  and  $\text{ind}'$ . It returns the sum of the size of these regions and their limits *i.e.* the smallest (resp. biggest) counter  $\text{cnt}_x$  (resp.  $\text{cnt}_y$ ) such that  $\Gamma_{\ell'}[w, \text{del}, \text{cnt}_x] = (\ell, \text{ind}_x)$  and  $\text{ind}_x \geq \text{ind}$  (resp.  $\Gamma_{\ell'}[w, \text{del}, \text{cnt}_y] = (\ell, \text{ind}_y)$  and  $\text{ind}_y \leq \text{ind}'$ ). We refer to these limits as *hole components*.

Finally, for the sake of verifiability, we need to keep track of these holes. So we add them to a hash table of lists AllHoles. AllHoles $[\ell']$  stores all the hole components at level  $\ell'$ . Note that, as we process levels increasingly, the target level  $\ell$  of components added to AllHoles $[\ell']$  increases similarly, and as among levels, entries are processed with increasing counter, counters  $\text{cnt}_x$  and  $\text{cnt}_y$  also increase. Actually, if  $(\ell^1, \text{cnt}_x^1, \text{cnt}_y^1)$  and  $(\ell^2, \text{cnt}_x^2, \text{cnt}_y^2)$  are two successive entries in AllHoles $[\ell']$ ,  $\ell^1 \leq \ell^2$  and  $\text{cnt}_x^1 \leq \text{cnt}_y^1 < \text{cnt}_x^2 \leq \text{cnt}_y^2$ . Hence, the server does not have to go through sorting to get a sorted list (which is important for computational complexity).

In the end, ProcessLevel returns the lists  $\mathcal{I}_\ell$  whose elements are either add entries for  $w$  in level  $\ell$  (together with some membership proof) or holes for  $w$  with target level  $\ell$ . The last element of this list is always the  $\perp$  entry with a VHT proof corresponding to the last counter value of the main loop (the smallest  $\text{cnt}$  for which Lookup returns  $\perp$ ).

**Proving and verifying holes.** The server has to prove to the client that he did not cheat when producing the holes. To do so, we could first use the verifiable hash table to show that the hole components limits are genuine del entries, which is what ProveHoles does. Unfortunately, this is not enough: for example the server could return two holes for which the intersection of components at a level  $\ell$  is not empty, which never happens if the search protocol is run fairly. He could also intentionally omit a del entry. Fortunately, we can use some very interesting properties of the holes to avoid any falsification of the results by the server, which we use in ProveHoles and VerifyHoles (Algorithm 6.24).

First, let us fix a level  $\ell$ . We will consider only hole components or del entries whose level is  $\ell$ . For such a del entry, there exists an add entry in a higher level  $\ell'$ . Hence it should belong to a hole component with target level  $\ell'$ . Said otherwise, every del entries belong to a hole, and if the client wants to check that the server rightly took into account every del entries, he should check that the hole components at level  $\ell$  span all its del entries. This is exactly what CheckAdjacency does, using the fact that hole components are *consecutive*.

In the previous paragraph, we noticed that the elements in AllHoles $[\ell]$  have components increasing in a very particular way. Let  $(\ell^1, \text{cnt}_x^1, \text{cnt}_y^1)$  be an element of AllHoles $[\ell]$ . The del entry with counter value  $\text{cnt}_y^1 + 1$  must be either  $\perp$ , or a del entry with target level  $\ell^* \geq \ell^1$  (because of the sorting key used in Rebuild). In the latter case, it necessarily is the start of a new hole component at level  $\ell$ . Hence, if  $(\ell^1, \text{cnt}_x^1, \text{cnt}_y^1)$  and  $(\ell^2, \text{cnt}_x^2, \text{cnt}_y^2)$  are two successive entries in AllHoles $[\ell]$ , we must have  $\ell^1 \leq \ell^2$  and  $\text{cnt}_x^1 \leq \text{cnt}_y^1 = (\text{cnt}_x^2 - 1) < \text{cnt}_y^2$ . This condition is checked at line 3 of CheckAdjacency. Testing that the server reached the last del entries of level  $\ell$  is done by verifying a non membership proof for counter  $\text{cnt}_{last} + 1$  at line 13, where  $\text{cnt}_{last}$  is the highest counter value encountered among the hole components of level  $\ell$ . The proof was produced by the server at line 10.

**The CheckResults algorithm.** The last thing the client has to do is to check the consistency of the result lists  $\mathcal{I}_\ell$  and holes. This is what CheckResults achieves, as described in Algorithm 6.25. It verifies three points: first that every add entry in  $\mathcal{I}_\ell$  has no matching del entry among the holes, then that the holes in  $\mathcal{I}_\ell$  are verified holes, and finally that all the add entries at level  $\ell$  were considered when searching.

**Algorithm 6.24** Prove and verify holes.ProveHoles(AllHoles)


---

```

1: Initialize  $\Pi_{\text{Holes}}$  as an empty table of lists.
2: for  $\ell = 0$  to  $L$  do
3:    $cnt_{last} \leftarrow -1$ 
4:   for all  $(\ell', cnt_x, cnt_y, \cdot, \cdot)$  in AllHoles[ $\ell$ ] in increasing order do
5:      $(entry_x, \pi_x) \leftarrow \text{Lookup}(\text{token}_\ell, \text{del}, cnt_x)$ 
6:      $(entry_y, \pi_y) \leftarrow \text{Lookup}(\text{token}_\ell, \text{del}, cnt_y)$ 
7:     Append  $((cnt_x, entry_x, \pi_x), (cnt_y, entry_y, \pi_y))$  (in that order) to  $\Pi_{\text{Holes}}[\ell]$ 
8:      $cnt_{last} \leftarrow cnt_y$ 
9:   end for
10:   $(entry_{last}, \pi_{last}) \leftarrow \text{Lookup}(\text{token}_\ell, \text{del}, cnt_{last} + 1)$   $\triangleright entry_{last} = \perp$ 
11:  Append  $\pi_{last}$  to  $\Pi_{\text{Holes}}[\ell]$ 
12: end for
13: return  $\Pi_{\text{Holes}}$ 

```

VerifyHoles( $\Pi_{\text{Holes}}$ )

```

1: Initialize AllHoles as an empty table of lists.
2: for  $\ell = 0$  to  $L$  do
3:    $cnt_{last} \leftarrow -1$ 
4:   for all  $((cnt_x, entry_x, \pi_x), (cnt_y, entry_y, \pi_y)) \in \Pi_{\text{Holes}}[\ell]$  do
5:      $hkey_x \leftarrow H_{\text{token}_\ell}(0||\text{del}||cnt_x)$ ,  $hkey_y \leftarrow H_{\text{token}_\ell}(0||\text{del}||cnt_y)$ 
6:     VHTVerify( $\sigma_{\text{VHT}_\ell}$ ,  $hkey_x$ ,  $entry_x, \pi_x$ ), VHTVerify( $\sigma_{\text{VHT}_\ell}$ ,  $hkey_y$ ,  $entry_y, \pi_y$ )
7:     Parse  $entry_x$  as  $(l_x^*, ind_x, c_{2x})$ , and  $entry_y$  as  $(l_y^*, ind_y, c_{2y})$ 
8:     if  $l_x^* \neq l_y^*$  return REJECT
9:     Append  $(l_x^*, cnt_x, cnt_y, ind_x, ind_y)$  to AllHoles[ $\ell$ ]
10:     $cnt_{last} \leftarrow cnt_y$ 
11:  end for
12:   $hkey_{last} \leftarrow H_{\text{token}_\ell}(0||\text{del}||cnt_{last} + 1)$ 
13:  VHTVerify( $\sigma_{\text{VHT}_\ell}$ ,  $hkey_{last}$ ,  $\perp$ ,  $\pi_{last}$ )
14:  CheckAdjacency(AllHoles[ $\ell$ ])
15: end for
16: return AllHoles

```

CheckAdjacency(AllHoles[ $\ell$ ])

```

1:  $cnt \leftarrow -1$ ,  $\ell^* \leftarrow 0$ 
2: for all  $(\ell', cnt_x, cnt_y, \cdot, \cdot) \in \text{AllHoles}[\ell]$  in increasing order do
3:   if  $\ell^* > \ell'$ ,  $cnt_x \neq cnt + 1$  or  $cnt_x > cnt_y$ 
4:     return REJECT
5:    $cnt \leftarrow cnt_y$ ,  $\ell^* \leftarrow \ell'$ 
6: end for
7: return ACCEPT

```

---

First, one must notice that, if the Search protocol is fairly executed by the server, one cannot find in  $\mathcal{I}_\ell$  two consecutive holes. Otherwise, it would say that the first hole is not maximal, unlike what SkipHole ensures. Hence, in CheckResults (line 15), we check that there are not two consecutive holes entries in  $\mathcal{I}_\ell$ . In the following, we will suppose that a hole is immediately followed by an add

entry in lists  $\mathcal{I}_\ell$ .

One of the key thing to check is that the document's index of successively considered entries strictly increases: in ProcessLevel, the server adds to  $\mathcal{I}_\ell$  either add entries retrieved using an incrementing counter or holes whose components limits indices must be larger than the previous add entry and smaller than the next one, by construction of SkipHole. This is checked at lines 10 and 19.

---

**Algorithm 6.25** CheckResults algorithm.

---

CheckResults( $\mathcal{I}_0, \dots, \mathcal{I}_L, \text{AllHoles}$ )

---

```

1:  $\mathcal{I} \leftarrow \emptyset$ 
2:  $\text{ind}_{last} \leftarrow -1$ 
3:  $\text{cnt} \leftarrow 0$ 
4: for  $\ell = 0$  to  $L$  do
5:   for all  $e \in \mathcal{I}_\ell$  do
6:     if  $e$  is  $(\text{entry}, \pi)$  then
7:        $\text{hkey}_x \leftarrow H_{\text{token}_\ell}(0||\text{add}||\text{cnt})$ 
8:        $\text{VHTVerify}(\sigma_{\text{VHT}_\ell}, \text{hkey}, \text{entry}, \pi)$ 
9:        $(w, \ell, \text{ind}, \text{add}, \text{cnt}) \leftarrow \text{ADec}(\text{esk}, (\ell, r_\ell), c_2)$   $\triangleright$  As entry has been verified, we know
       that ADec returns keyword  $w$ , target level  $\ell$  and operation add.
10:      if  $\text{ind}_{last} \geq \text{ind}$ 
11:        return REJECT
12:       $\text{ind}_{last} \leftarrow \text{ind}, \text{cnt}++, \mathcal{I} \leftarrow \mathcal{I} \cup \{\text{ind}\}$ 
13:      else if  $e$  is a hole Hole then
14:        if previous entry was a hole
15:          return REJECT
16:         $\text{sum} \leftarrow 0, \text{ind}_{max} \leftarrow \text{ind}_{last}$ 
17:        for all  $h \in \text{Hole}$  do
18:          Parse  $h$  as  $(\ell', \ell^*, \text{cnt}_x, \text{cnt}_y, \text{ind}_x, \text{ind}_y)$ 
19:          if  $\ell^* \neq \ell, h \notin \text{AllHoles}[\ell']$  or  $\text{ind}_x \leq \text{ind}_{last}$ 
20:            return REJECT
21:           $\text{ind}_{max} \leftarrow \max(\text{ind}_{max}, \text{ind}_y), \text{sum} \leftarrow \text{sum} + \text{cnt}_y - \text{cnt}_x + 1$ 
22:        end for
23:         $\text{ind}_{last} \leftarrow \text{ind}_{max}, \text{cnt} \leftarrow \text{cnt} + \text{sum} + 1$ 
24:      else  $\triangleright e$  is the proof  $\pi_{last}$ 
25:         $\text{hkey}_{last} \leftarrow H_{\text{token}_\ell}(0||\text{add}||\text{cnt})$ 
26:         $\text{VHTVerify}(\sigma_{\text{VHT}_\ell}, \text{hkey}, \perp, \pi_{last})$ 
27:        if  $e$  is not the last element of  $\mathcal{I}_\ell$ 
28:          return REJECT
29:        end if  $\triangleright$  Test value of  $e$ 
30:      end for  $\triangleright$  Loop over  $\mathcal{I}_\ell$ 
31:    end for  $\triangleright$  Loop over the levels
32:  return  $\mathcal{I}$ 

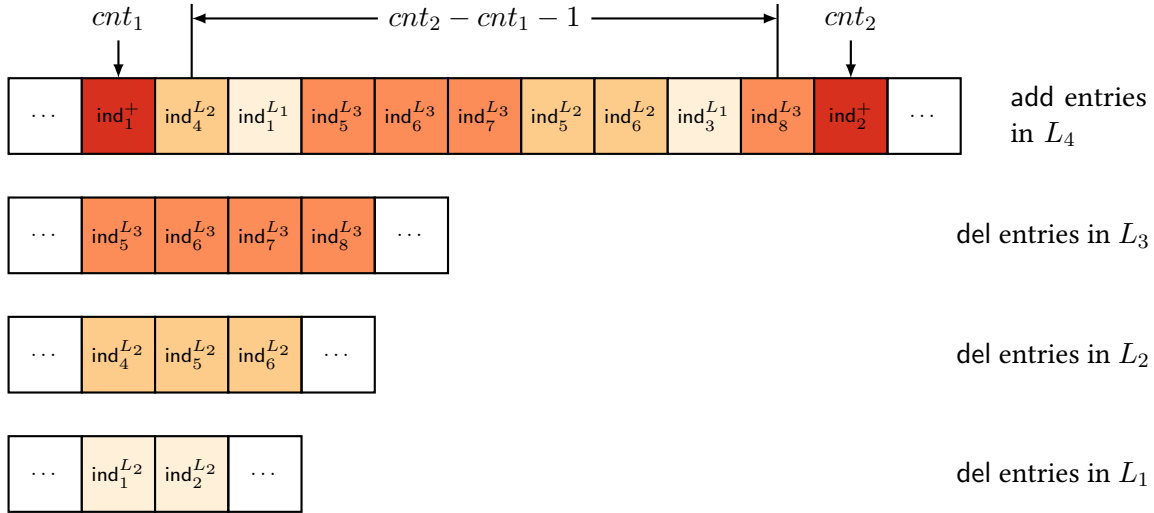
```

---

CheckResults would also reject when holes are incorrect, *i.e.* when their target level is not  $\ell$ , or if they were not registered in AllHoles. Note that if a hole  $h$  is in AllHoles, it has been proved correct by a previous ProveHole call, and we do not have to show that its components were correctly formed and that their limits were genuine entries.



The algorithm also checks that the add entries are correct. In particular, the counter value used to retrieve the hkey is not provided by the server, but recomputed by the client: it is incremented when the current element  $e$  is an add entry, and, if  $e$  is a hole, increased by its width. By doing so, and relying on the increasing document indices, we ensure that the holes width is correct, *i.e.* that the width of a hole is equal to the difference of counter values of neighboring add entries. Figure 6.4 illustrates this point.



**Figure 6.4** – Example of what CheckResults checks. In this example,  $\mathcal{I}_{L_4}$  contains the following sequence: add entry with counter  $cnt_1$  and index  $ind_1^+$ , a hole whose components are at levels  $L_3$ ,  $L_2$  and  $L_1$ , and another add entry with counter  $cnt_2$  and index  $ind_2^+$ . The upper row represents the add entries for keyword  $w$  at level  $L_4$ . In orange, beige and sand are the ones with a matching entry in the hole. The red entries are the add entries in  $\mathcal{I}_{L_4}$

**Merging the VHT in the level table and de-amortization.** If one uses Section 6.2.2 as the VHT instantiation  $\Theta$ , one would notice that the contents of the hash tables, *i.e.* the entries created by EncodeEntry, are authenticated ciphertexts (at least for the member  $c_2$ ) – essentially a ciphertext associated with a MAC – which are then MACed with their position in the table. We could avoid this second MAC by adding the position in the table to the associated data of the authenticated encryption, and including this position as a member of the entry. All the algorithms described previously would not change, except that the VHTVerify calls would be replaced by the verification that the decryption procedure ADec did not return REJECT when verifying a membership proof.

Moreover, this modification (including the position index in the associated data) would essentially be transparent to the oblivious sort algorithm. Hence, we could apply the de-amortization as in [SPS14], without caring about any extra cost induced by the verified hash table. Using de-amortization, the average case complexity becomes the worst-case complexity: the update operation takes  $\mathcal{O}(\log^2 N)$  time and induces  $\mathcal{O}(\log N)$  bandwidth, in the worst case.

If we were to use another instantiation for  $\Theta$ , we would have to make sure we can use de-amortization on this instantiation to avoid unacceptable worst-case complexity.

### 6.5.5 Soundness proof of Verif-SPS

The soundness relies both on the verifiable hash table and on the authenticated encryption scheme, it is easy to see that using a hybrid argument that will successively suppose the complete soundness of the VHT, and unforgeability of the authenticated encryption scheme, we will be able to prove the soundness of Verif-SPS. Yet, in the proof, we will also have to suppose that SPS is also fully correct. More formally, we have the following theorem.

**Theorem 6.8** (Soundness of Verif-SPS). *For every adversary  $A$ , there are adversaries  $B$ ,  $C$  and  $D$  such that*

$$\text{Adv}_{\text{Verif-SPS},A}^{\text{SSE-snd}}(\lambda) \leq N \cdot \text{Adv}_{\Theta,B}^{\text{VHT-snd}}(\lambda) + \text{Adv}_{\text{AEnc},C}^{\text{ae}}(\lambda) + \text{Adv}_{\text{SPS},D}^{\text{SSE-corr}}(\lambda)$$

We can now give the full proof of the soundness of the Verif-SPS scheme. We will use the following, two step, strategy:

1. We create a reduction, using hybrids, to a derivative of the game SSESOUND were the values given by the adversary are ‘sound’, *i.e.* are either correctly verified or fail verification. For example, a value issued from a verifiable hash table will never be a successful forgery.
2. We show that this game cannot be won by any adversary.

#### 6.5.5.1 Reduction

We construct 4 games  $G_0$  to  $G_3$ ,  $G_0$  being (almost) the original SSESOUND security game. We will go from  $G_0$  to  $G_3$  by successively assuming the soundness of the verifiable hash table  $\Theta$ , the authenticity of the encryption scheme AEnc, and finally the correctness of Verif-SPS.

**Game  $G_0$ .**  $G_0$  is exactly the game  $\text{SSESOUND}_{\text{Verif-SPS}}^A$ , up to one difference: for every VHT or authenticated encryption verification, if a forgery actually succeeded,  $G_0$  will immediately set the flag win to true. More formally, each call to  $\text{VHTVerify}(\sigma_{\text{VHT}_\ell}, \text{hkey}, \text{entry}, \pi)$  is followed by the pseudo-code

```

if  $\mathbf{T}_\ell[\text{hkey}] \neq \text{entry}$ 
  and  $\text{VHTVerify}(\sigma_{\text{VHT}_\ell}, \text{hkey}, \text{entry}, \pi) = \text{ACCEPT}$ 
  win  $\leftarrow$  true

```

and the decryptions  $\text{tuple} \leftarrow \text{ADec}(\text{esk}, (\ell, r_\ell), c_2)$  by

```

if  $\text{tuple} \neq \text{REJECT}$  and  $c_2$  is not an encryption of  $\text{tuple}$ 
  win  $\leftarrow$  true

```

The adversary  $A$  has more chances to win  $G_0$  than the original SSESOUND game and we have

$$\mathbb{P}[\text{SSESOUND}_{\text{Verif-SPS}}^A(1^\lambda) = 1] \leq \mathbb{P}[G_0(1^\lambda) = 1]$$

**Game  $G_1$ .** In game  $G_1$ , all calls to  $\text{VHTVerify}(\sigma_{\text{VHT}_\ell}, \text{hkey}, \text{entry}, \pi)$  (including the pseudo code added in  $G_0$ ) are replaced by the following:

```

if  $\mathbf{T}_\ell[\text{hkey}] \neq \text{entry}$ 
  return REJECT

```

**return** *entry*

Said otherwise, we suppose that all the VHTVerify calls perform as expected, rejecting forgery attempts.

It is important to notice that, at every update, exactly one level is rebuilt. So, let us consider sub-hybrids  $G_0^0, G_0^1, \dots, G_0^N$ , such that for all  $i$ , in  $G_0^i$ , all the tables rebuilt before the  $i$ -th update (included) use the upper pseudo-code for verification, and all the other tables rebuilt after this update use the regular verification procedure. We have that, for all  $1 \leq i \leq N$ , there exists an adversary  $B_i$  such that

$$\mathbb{P}[G_0^{i-1}(1^\lambda) = 1] - \mathbb{P}[G_0^i(1^\lambda) = 1] \leq \text{Adv}_{\Theta, B_i}^{\text{VHT-snd}}(\lambda).$$

$B_i$  tries to attack the soundness of the table rebuild at the  $i$ -th update. Note that  $B_i$  does not make any call to Update in the game VHTSOUND (the tables are static), and the number of calls to Challenge is upper bounded by the number of tokens sent by  $A$ .

Summing the probabilities, as  $G_0^0 = G_0$  and  $G_0^N = G_1$ , we have that there exists an adversary  $B$  such that

$$\mathbb{P}[G_1(1^\lambda) = 1] - \mathbb{P}[G_0(1^\lambda) = 1] \leq N \cdot \text{Adv}_{\Theta, B}^{\text{VHT-snd}}(\lambda).$$

**Game  $G_2$ .** In game  $G_2$ , all the tentative forgeries of authenticated ciphertexts will fail: the adversary has to give to the game valid ciphertext or these will be rejected.

Notice that, as we have from  $G_1$  that all the entries issued from the verifiable hash table are valid, in particular, their field  $c_2$  has to be valid too, and hence decrypt correctly, without rejection. It implies that the adversary can win  $G_1$  by providing forged ciphers in the Rebuild protocol. As a consequence, in  $G_2$ , we remove all the code added in  $G_0$  after calls to ADec in the Search protocol, without modifying the adversary's success probability. In the rest of the paragraph, we will focus only on the Rebuild protocol.

When running Rebuild, when decrypting a tuple, the game will always know what result he should expect: the algorithms are deterministic, except for the encryption but it does not influence the order the tuples are treated. So, we modify the game  $G_1$  the following way to give  $G_2$ : in  $G_2$  all calls to ADec are followed by:

```

if ( $w, \ell^*$ , ind, op, cnt) has not been encrypted with the same authenticated data as the one used
for decryption
return REJECT

```

We also do this in the o-sort protocol, which uses an authenticated encryption scheme (and associated data too).

In  $G_2$ , the game accepts to decrypt only ciphertexts that have been generated by the game itself with the additional data that are used for the decryption. We are exactly in the case of the authentication security game for encryption schemes and hence, there exists an adversary  $C$  such that

$$\mathbb{P}[G_2(1^\lambda) = 1] - \mathbb{P}[G_1(1^\lambda) = 1] \leq \text{Adv}_{\text{AEnc}, C}^{\text{ae}}(\lambda)$$

$C$  sees all the encryptions generated by the Rebuild algorithms (including the o-sort protocol), and tries to forge at most once per Rebuild call (remember that the protocol halts as soon as it gets REJECT) *i.e.* at most  $N$  times.

We also recall that, as we supposed o-sort to be secure against malicious adversaries, when enumerating the entries of  $\hat{\mathbf{B}}$ , they come in sorted order, and are the same as the entries of the input table.

**Game  $G_3$ .** In game  $G_3$ , we will suppose that all the lookups behave normally: when looking for keyword  $w$ , with operation  $op$  and counter  $cnt$  at level  $\ell$ ,  $\text{Lookup}(\text{token}_\ell, op, cnt)$  will actually return an entry for keyword  $w$ . Still another way to see it is to say that the schemes behave exactly as expected on the conceptual data structure  $\Gamma$ . In particular, it implies that there is no collision when computing  $\text{token}_\ell$  or  $hkey$ . This is ensured by the correctness of the original SPS construction, and we have that there exists an adversary  $D$  such that

$$\mathbb{P}[G_3(1^\lambda) = 1] - \mathbb{P}[G_2(1^\lambda) = 1] \leq \text{Adv}_{\text{SPS}, D}^{\text{SSE-corr}}(\lambda)$$

Finally, when we sum up the contribution of all the games, we infer that

$$\text{Adv}_{\text{Verif-SPS}, A}^{\text{SSE-snd}}(\lambda) \leq N \cdot \text{Adv}_{\Theta, B}^{\text{VHT-snd}}(\lambda) + \text{Adv}_{\text{AEnc}, C}^{\text{ae}}(\lambda) + \text{Adv}_{\text{SPS}, D}^{\text{SSE-corr}}(\lambda)$$

In the next section, we will show that  $\mathbb{P}[G_3 = 1] = 0$ .

### 6.5.5.2 Soundness of $G_3$

Without loss of generality, we can suppose that the adversary never gives REJECT to the game  $G_3$ : when it is the case, the current procedure would immediately halt and return REJECT, and the flag `win` will not be set to `true`. As a consequence, in this section, we will suppose that none of the procedures `Search` or `Update` return REJECT. It implies that, in  $G_3$ , the entries given by the server to the client are genuine, and that ciphertexts decrypted by the client were previously encrypted with the same authenticated data.

The first thing we want to prove is that the adversary cannot corrupt the table  $\mathbf{T}_\ell$ : its content always reflects the conceptual data structure  $\Gamma_\ell$ .

**Proposition 6.9.** *At anytime in the execution of the game, if  $(\ell^*, \text{ind}) = \Gamma_\ell[w, op, cnt]$  then, in  $G_3$ ,  $\text{Lookup}(\text{token}_\ell, op, cnt) = (\ell^*, \text{ind}, \dots)$  with  $\text{token}_\ell = F(k_\ell, h(w))$*

*Proof.* Because of the timestamp  $r_\ell$  and the flags used during the rebuilding, ciphertexts generated cannot be replayed. More precisely, all the ciphertexts generated for level  $\ell$  before the reconstruction cannot be reused because of the new value of the timestamp  $r_\ell$  that was incremented upon rebuild. Then, ciphertexts generated before the first call to `o-sort` cannot be reused in place of the ciphertexts generated between the two calls to `o-sort` which themselves cannot be reused later: the firsts are encrypted using  $(\ell, r_\ell, 0)$  as associated data, while the seconds use  $(\ell, r_\ell, 1)$  and the lasts only  $(\ell, r_\ell)$ .

We conclude with the correctness of the oblivious sort algorithm: at the end of the protocol's execution, the entries are always correctly sorted in  $\mathbf{T}_\ell$ , unless it returned REJECT.  $\square$

Now, we can focus on the `Search` protocol. We only treat the sublinear case, as the soundness of the basic construction is immediate, and we will proceed level per level: the result set is the (disjoint) union of the results produced by each level. For lemmas will be proven:

- $\text{AllHoles}[\ell]$  forms a partition of the del entries in level  $\ell$ . Moreover, in each hole component, entries are consecutive.
- At level  $\ell$ , all the add entries returned by the server in  $\mathcal{I}_\ell$  match no del entry in lower levels.
- At level  $\ell$ , every add entry in a hole defined by  $\mathcal{I}_\ell$  match a del entry in a lower level.
- At level  $\ell$ , every add entry has been returned by the server, either as a non-deleted entry, or an entry belonging to a hole.

**Lemma 6.10.** *Let  $\ell$  be a level, and AllHoles the hole table sent by the server after a search request with keyword  $w$ . Let*

$$((cnt_x^i, entry_x^i, \pi_x^i), (cnt_y^i, entry_y^i, \pi_y^i))_{i=0}^s$$

be the list AllHoles $[\ell]$ .

Then  $(\{\Gamma_\ell[w, \text{del}, cnt]\}_{cnt=cnt_x^i}^{cnt_y^i})_{i=0}^s$  is a partition of

$$\{\Gamma_\ell[w, \text{del}, cnt]\}_{cnt=0}^{cnt_{last}}$$

where  $cnt_{last}$  is the largest integer such that  $\Gamma_\ell[w, \text{del}, cnt_{last} + 1] \neq \perp$

*Proof.* This lemma entirely relies on the procedure CheckAdjacency. It checks that the hole components in AllHoles $[\ell]$  are consecutive: if  $(\ell^*, cnt_x, cnt_y, ind_x, ind_y)$  and  $(\ell'^*, cnt'_x, cnt'_y, ind'_x, ind'_y)$  are successive elements in AllHoles $[\ell]$ , then  $cnt_x \leq cnt_y$ ,  $cnt'_x \leq cnt'_y$  and  $cnt'_x = cnt_y + 1$ . Hence, when CheckAdjacency accepts, we know that the sets  $\{\Gamma_\ell[w, \text{del}, cnt]\}_{cnt=cnt_x^i}^{cnt_y^i}$  are pairwise disjoint.

This is ensured for  $cnt_x$  starting at 0: if the first element of AllHoles $[\ell]$  has not  $cnt_x = 0$ , the test fails. Also, in VerifyHoles, we checked that, if  $cnt_{last}$  is the value of  $cnt_y$  for the last element of AllHoles $[\ell]$ , it has to be that  $\Gamma_\ell[w, \text{del}, cnt_{last}] \neq \perp$  while  $\Gamma_\ell[w, \text{del}, cnt_{last} + 1] = \perp$ . The verification spanned all the possible counter values for del entries at level  $\ell$ . Thus,

$$(\{\Gamma_\ell[w, \text{del}, cnt]\}_{cnt=cnt_x^i}^{cnt_y^i})_{i=0}^s$$

is a partition of  $\{\Gamma_\ell[w, \text{del}, cnt]\}_{cnt=0}^{cnt_{last}}$ .  $\square$

**Lemma 6.11.** *Let  $\ell$  be a level and  $\mathcal{I}_\ell$  the result list returned by the server for this level after a search request with keyword  $w$ . Let  $(entry, \pi)$  be an add element of  $\mathcal{I}_\ell$ , with entry decrypting to  $(w, \ell, ind, add, cnt)$ . If Search did not return REJECT, there is no del entry matching  $(ind, \ell)$  in any level  $\ell' < \ell$ : for all  $c$ ,  $\ell' < \ell$ ,  $\Gamma_{\ell'}[w, \text{del}, c] \neq (ind, \ell)$ .*

*Proof.* For this lemma, we use the guaranties offered by CheckResults, and in particular on the fact that the inner variable  $ind_{last}$  strictly increases when enumerating the elements of  $\mathcal{I}_\ell$ .

As a consequence, all the hole components  $(\ell', \ell, cnt_x, cnt_y, ind_x, ind_y)$  in  $\mathcal{I}_\ell$  encountered before *entry* are such that  $ind_y < ind$  (CheckResults would have failed otherwise), and all the hole components  $(\ell', \ell, cnt_x, cnt_y, ind_x, ind_y)$  in  $\mathcal{I}_\ell$  encountered after *entry* are such that  $ind_x > ind$ .

We conclude using Lemma 6.10: if there is  $cnt_{del}$  and  $\ell' < \ell$  such that  $\Gamma_{\ell'}[w, \text{del}, c] = (ind, \ell)$ , there is a hole component  $(\ell', \ell, cnt_x, cnt_y, ind_x, ind_y)$  with  $ind_x \leq ind \leq ind_y$ .  $\square$

**Lemma 6.12.** *Let  $\ell$  be a level and  $\mathcal{I}_\ell$  the result list returned by the server for this level after a search request with keyword  $w$ . Let Hole be a hole in  $\mathcal{I}_\ell$ ,  $(entry, \pi)$  and  $(entry', \pi')$  the two add entries flanking Hole in  $\mathcal{I}_\ell$ . *entry* (resp. *entry'*) decrypts to  $(w, \ell, ind, add, cnt)$  (resp.  $(w, \ell, ind', add', cnt')$ ). If Hole is the first element of  $\mathcal{I}_\ell$ , we set  $cnt = 0$ , and if it is the last element of  $\mathcal{I}_\ell$ , we set  $cnt$  to the smallest integer such that  $\Gamma_\ell[w, \text{add}, cnt] = \perp$ .*

*Then, for every  $cnt < c < cnt'$ ,  $(ind_c, \ell) = \Gamma_\ell[w, \text{add}, c]$ , there exists  $\ell' < \ell$  and  $c_{del}$  such that  $(ind_c, \ell) = \Gamma_{\ell'}[w, \text{del}, c_{del}]$ . There also is a hole component  $h = (\ell', \ell, cnt_x, cnt_y, ind_x, ind_y) \in \text{Hole}$  with  $cnt_x \leq c_{del} \leq cnt_y$ ,  $ind_y < ind < ind_x$ .*

*Proof.* Let us consider the set

$$\Delta = \bigcup_{h \in \text{Hole}} \{\Gamma_{\ell'}[w, \text{del}, cnt] \mid h = (\ell', \ell, cnt_x, cnt_y, ind_x, ind_y) \text{ and } cnt_x \leq cnt \leq cnt_y\}$$

Because we know that the elements of the union are pairwise disjoint, and because we checked the size of the hole by summing the difference  $cnt_y - cnt_x + 1$  for every component, we have that

$$|\Delta| = cnt' - cnt - 1$$

which is the number of  $c$  such that  $cnt < c < cnt'$ .

Let  $h = (\ell', \ell, cnt_x, cnt_y, ind_x, ind_y)$  be a hole component. As we saw earlier, we know that  $T_{\ell'}$  reflects the conceptual table  $\Gamma_{\ell'}$ , for each  $cnt_x \leq c \leq cnt_y$ ,  $\Gamma_{\ell'}[w, del, c] = (ind_c, \ell)$ . We also know that every del entry in  $\Gamma_{\ell'}[w, del, c] = (ind_c, \ell^*)$  (no condition on  $c$ ) has a matching entry  $\Gamma_{\ell^*}[w, add, c_{add}] = (ind_c, \ell^*)$ . Given Lemma 6.10, as the hole components at level  $\ell'$  form a partition of successive del entries at level  $\ell'$ , it must be that for all  $cnt_x \leq c \leq cnt_y$ ,

$$ind < ind_x \leq ind_c \leq ind_y < ind',$$

where  $\Gamma_{\ell'}[w, del, c] = (ind_c, \ell)$ .

This gives us that there are  $cnt' - cnt - 1$  distinct add entries at level  $\ell$  whose indices are strictly comprised between  $ind$  and  $ind'$ . From  $entry$  and  $entry'$ , we also know that there are exactly  $cnt' - cnt - 1$  distinct add entries at level  $\ell$ ,  $\Gamma_{\ell}[w, add, c] = (ind_c, \ell)$  with  $cnt < c < cnt'$ , and  $ind < ind_c < ind'$ . These have to be the same ones, proving the lemma.  $\square$

**Lemma 6.13.** *Let  $\ell$  be a level and  $\mathcal{I}_{\ell}$  the result list returned by the server for this level after a search request with keyword  $w$ . For every  $cnt$  such that  $\Gamma_{\ell}[w, add, cnt] = (ind, \ell)$ , either there is an entry  $entry \in \mathcal{I}_{\ell}$  decrypting to  $(w, \ell, ind, add, cnt)$  or there is a hole  $Hole \in \mathcal{I}_{\ell}$  with a component  $h = (\ell', \ell, cnt_x, cnt_y, ind_x, ind_y)$  with  $ind_x \leq ind$ , and  $ind \leq ind_y$ .*

*Proof.* Let  $cnt_{last}$  be the last counter value encountered when enumerating  $\mathcal{I}_{\ell}$ : if its last element is an add entry decrypting to  $(w, \ell, ind, add, cnt)$ ,  $cnt_{last} = cnt$ , if its last element is a hole  $(\ell', \ell, cnt_x, cnt_y, ind_x, ind_y)$ ,  $cnt_{last} = cnt$ .

CheckResults ensured that  $\Gamma_{\ell}[w, add, cnt_{last}] \neq \perp$  and  $\Gamma_{\ell}[w, add, cnt_{last} + 1] = \perp$ . It means that the add entries' count at level  $\ell$  span the interval  $[0, cnt_{last}]$ .

Let  $c$  be in this interval. Suppose that, in  $\mathcal{I}_{\ell}$ , there is no entry decrypting to  $(w, \ell, ind_c, add, c)$ . We also know that in  $\mathcal{I}_{\ell}$ , the value of the counter of add entries strictly increases. Hence, there is two counters  $cnt < c < cnt'$  satisfying the following conditions: there is two add entries  $entry$  and  $entry'$  in  $\mathcal{I}_{\ell}$  decrypting to  $(w, \ell, ind, add, cnt)$  and  $(w, \ell, ind', add, cnt')$ ,  $entry'$  is the add entry following  $entry$  in  $\mathcal{I}_{\ell}$ .

$entry$  and  $entry'$  cannot be immediately successive in  $\mathcal{I}_{\ell}$ : if that were the case, we would have  $cnt' = cnt + 1$  and one of  $entry$  or  $entry'$  would decrypt to  $(w, \ell, ind_c, add, c)$ . So, there is the hole  $Hole$  between these in  $\mathcal{I}_{\ell}$ , and with Lemma 6.12, we conclude that there is a hole component  $h = (\ell', \ell, cnt_x, cnt_y, ind_x, ind_y)$  with  $ind_x \leq ind_c \leq ind_y$ .  $\square$

With these lemmas, we can now easily show the perfect soundness of the hybrid of game  $G_3$ .

*Proof of Theorem 6.8.* As before, we proceed level per level. So let  $\ell$  be a level, and  $cnt$  such that

$$\Gamma_{\ell}[w, add, cnt] \neq \perp.$$

Lemma 6.13 tells us that there is either a corresponding entry  $entry$  or a correspond hole. In case of an add entry, Lemma 6.11 guaranties us that  $\Gamma_{\ell}[w, add, cnt]$  has not been deleted in a lower level. In case of a hole, Lemma 6.12 implies that  $\Gamma_{\ell}[w, add, cnt]$  as a matching del entry in a lower level.

In the sublinear Search algorithm, the result set is constructed from the add entries of the  $\mathcal{I}_\ell$  lists, which correspond to all the  $\Gamma_\ell[w, \text{add}, \text{cnt}]$  that have not been deleted. If Search did not return REJECT in game  $G_3$ , it returned the correct result set  $\text{DB}(w)$ .

$$\mathbb{P}[G_3(1^\lambda) = 1] = 0.$$

Combined with the reduction of the previous section

$$\text{Adv}_{\text{Verif-SPS},A}^{\text{SSE-snd}}(\lambda) \leq N \cdot \text{Adv}_{\Theta,B}^{\text{VHT-snd}}(\lambda) + \text{Adv}_{\text{AEnc},C}^{\text{ae}}(\lambda) + \text{Adv}_{\text{SPS},D}^{\text{SSE-corr}}(\lambda).$$

□

### 6.5.6 Complexity

The complexity of this construction is very similar to the one of the original SPS scheme. The update communication complexity is unchanged: during the updates, the only additional elements sent between the client and the server are the tags of the authenticated ciphertexts (the active security of the oblivious sort relies only on them). During the search protocols, the server will send the proofs corresponding to every hole component, and as there are at most  $n_w$  holes, consisting of  $\log N$  components, the communication complexity of the search grows from  $\mathcal{O}(n_w)$  to  $\mathcal{O}(n_w \cdot \log N)$ .

For the time complexity, we can show that, if the VHT is instantiated using the construction of Section 6.2.2, search has complexity  $\mathcal{O}(a_w + \log^2 N)$  (instead of  $\mathcal{O}(a_w + \log N)$ ) where  $a_w$  is the number of entries matching the searched keyword in the database for the basic scheme, and  $\mathcal{O}(n_w \log^3 N)$  for the sublinear scheme (the asymptotic complexity is unchanged in this case), and update has an amortized update time of  $\mathcal{O}(\log^2 N)$ , and  $\mathcal{O}(N \log N)$  in the worst-case. As in the original scheme, we can use de-amortization techniques to make update time  $\mathcal{O}(\log^2 N)$  in the worst case.

#### 6.5.6.1 Complexity of the basic scheme

Let  $T^{\text{build}}(n)$  be the complexity of VHTSetup for an input table of size  $n$ ,  $T_\epsilon^{\text{prove}}(n)$  (resp.  $T_\perp^{\text{prove}}(n)$ ) the complexity of VHTGet, *i.e.* the proving complexity, when queried on a key in the table (resp. a non-member key), and  $T_\epsilon^{\text{check}}(n)$  (resp.  $T_\perp^{\text{check}}(n)$ ) the complexity of VHTVerify, *i.e.* the verification complexity, for a proof of an element in the table (resp. a non member element).

The Rebuild (without the VHT) algorithm takes time  $\mathcal{O}(2^\ell \ell)$  when  $\ell$  is the empty level to be filled, as the bottleneck phase is the oblivious sorting algorithm [GM11]. Computing the VHT induces an extra  $T^{\text{build}}(2^\ell)$  cost, as well as using authenticated encryption and the additional sanity checks. Hence, the worst case complexity of Update is  $\mathcal{O}(N \log N) + T^{\text{build}}(N)$  and  $\mathcal{O}(\log^2 N) + \sum_{\ell=0}^{L-1} T^{\text{build}}(2^\ell)/N = \mathcal{O}(\log^2 N + \frac{\log N}{N} T^{\text{build}}(N))$  amortized time, as each level  $\ell$  is rebuilt every  $2^\ell$  updates, each rebuild being of  $\mathcal{O}(\ell \cdot 2^\ell) + T^{\text{build}}(2^\ell)$  computational complexity.

The basic Search algorithm took time  $\mathcal{O}(\alpha + \log N)$  in the passive adversary setting of [SPS14], where  $\alpha$  is the number of times the keyword was added to the database. In our case, it increases to  $\mathcal{O}(\alpha(T_\epsilon^{\text{prove}}(n) + T_\epsilon^{\text{check}}(n)) + \log N(T_\perp^{\text{prove}}(n) + T_\perp^{\text{check}}(n)))$ . To be more precise, the server needs  $\mathcal{O}(\alpha T_\epsilon^{\text{prove}}(n) + \log N)$  to find through all entries with keyword  $w$  and get the associated proof, and for each level, it needs to produce two proofs for  $\perp$  (one for an add entry, the other for a del entry), which takes  $\mathcal{O}(\log N \cdot T_\perp^{\text{prove}}(n))$  time total.

Hence, using the instantiation of Section 6.2.2, we end up with  $\mathcal{O}(\alpha + \log^2 N)$  search time and  $\mathcal{O}(\log^2 N)$  amortized update time.

### 6.5.6.2 Complexity of the Sublinear Scheme

The sublinear construction reuses the same Rebuild as the basic construction. Hence, we directly have that Update take  $\mathcal{O}\left(\log^2 N + \frac{\log N}{N} T^{\text{build}}(N)\right)$  amortized time, and  $\mathcal{O}(N \log N) + T^{\text{build}}(N)$  in the worst case, using the conclusions of the previous section. We will focus here on the Search algorithm (Algorithms 6.22, 6.23, 6.24, and 6.25).

First, let us study ProcessLevel Namely, the server sends to the client  $L$  lists  $\mathcal{I}_\ell$  containing add entries and holes. There are exactly  $m$  such add entries total, and we know that a hole entry must be immediately followed by an add entry, and there are at most  $m$  holes in the lists  $\mathcal{I}_\ell$ . To ensure an add entry has no associated del entry, the server has to perform an unsuccessful binary search in every level, taking  $\mathcal{O}(\log^2 N)$  time. To compute a hole, SkipHole performs a binary search on the add entries of the current level  $\ell$ , and hence calls DeletedSum  $\mathcal{O}(\log N)$  times. Then, DeletedSum goes through all the lower levels and itself performs a binary search, taking  $\mathcal{O}(\log^2 N)$ . Finally, for each level, ProcessLevel produces a proof for a key not present in the VHT, taking  $\mathcal{O}(\log N)$ . If we sum all these contributions, ProcessLevel takes time  $\mathcal{O}(m \log^3 N)$ .

ProveHoles enumerates the hole components. As there are at most  $m$  holes, there are at most  $m \log N$  hole components. For each of these components, ProveHoles generate a VHT proof in constant time (components limits are real entries in the level). It also generates a “not found” proof for every level in  $\mathcal{O}(\log N)$ . ProveHoles takes time  $\mathcal{O}(m \log N \cdot T_\epsilon^{\text{prove}}(N) + \log N \cdot T_\perp^{\text{prove}}(N))$ .

On the verifier’s side, VerifyHoles verifies  $\mathcal{O}(m \log N)$  of these proofs and calls CheckAdjacency, which itself goes through all the  $\mathcal{O}(m \log N)$  hole components. VerifyHoles also verifies the  $\mathcal{O}(\log N)$  non-membership proofs in  $\mathcal{O}(T_\perp^{\text{check}}(N))$  time. Finally, CheckResults also uses the hole components and  $\mathcal{I}_\ell$ ’s add entries sequentially, each of them being processed in constant time.

The total verification time is  $\mathcal{O}(m \log N \cdot T_\epsilon^{\text{check}}(N) + \log N \cdot T_\perp^{\text{check}}(N))$ , and the total time complexity of the search algorithm is

$$\mathcal{O}(m \log^3 N + m \log N (T_\epsilon^{\text{prove}}(N) + T_\epsilon^{\text{check}}(N)) + \log N (T_\perp^{\text{prove}}(N) + T_\perp^{\text{check}}(N))).$$

Instantiated with the static VHT of Section 6.2.2, it gives us a search complexity of  $\mathcal{O}(m \log^3 N)$ .

## References

- [AKST14] Daniel Apon, Jonathan Katz, Elaine Shi, and Aishwarya Thiruvengadam. *Verifiable Oblivious Storage*. In: *PKC 2014*. Ed. by Hugo Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 131–148 (cit. on p. 130).
- [BEG+91] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. *Checking the Correctness of Memories*. In: *32nd FOCS*. IEEE Computer Society Press, Oct. 1991, pp. 90–99 (cit. on p. 120).
- [BFP16] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. *Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security*. Cryptology ePrint Archive, Report 2016/062. <http://eprint.iacr.org/2016/062>. 2016 (cit. on pp. ix, 11, 13, 119).
- [CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. 10, 49, 56, 73, 83, 99, 129, 163, 170, 171).



- [CL02] Jan Camenisch and Anna Lysyanskaya. *Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials*. In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 61–76 (cit. on p. 122).
- [DNRV09] Cynthia Dwork, Moni Naor, Guy N. Rothblum, and Vinod Vaikuntanathan. *How Efficient Can Memory Checking Be?* In: *TCC 2009*. Ed. by Omer Reingold. Vol. 5444. LNCS. Springer, Heidelberg, Mar. 2009, pp. 503–520 (cit. on p. 120).
- [GM11] Michael T. Goodrich and Michael Mitzenmacher. *Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation*. In: *ICALP 2011, Part II*. Ed. by Luca Aceto, Monika Henzinger, and Jiri Sgall. Vol. 6756. LNCS. Springer, Heidelberg, July 2011, pp. 576–587 (cit. on pp. 59, 71, 130, 133, 147).
- [MTA16] Jeremy Maitin-Shepard, Mehdi Tibouchi, and Diego F Aranha. “Elliptic Curve Multiset Hash”. In: *The Computer Journal* 60.4 (2016), pp. 476–490 (cit. on pp. 33, 129).
- [NR05] Moni Naor and Guy N. Rothblum. *The Complexity of Online Memory Checking*. In: *46th FOCS*. IEEE Computer Society Press, Oct. 2005, pp. 573–584 (cit. on p. 120).
- [PT08] Charalampos Papamanthou and Roberto Tamassia. *Time and Space Efficient Algorithms for Two-Party Authenticated Data Structures*. In: *ICICS 07*. Ed. by Sihang Qing, Hideki Imai, and Guilin Wang. Vol. 4861. LNCS. Springer, Heidelberg, Dec. 2008, pp. 1–15 (cit. on p. 120).
- [PTT09] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. *Cryptographic Accumulators for Authenticated Hash Tables*. Cryptology ePrint Archive, Report 2009/625. <http://eprint.iacr.org/2009/625>. 2009 (cit. on pp. 120, 122, 129).
- [RBB03] Phillip Rogaway, Mihir Bellare, and John Black. “OCB: A block-cipher mode of operation for efficient authenticated encryption”. In: *ACM Transactions on Information and System Security (TISSEC)* 6.3 (2003), pp. 365–403 (cit. on p. 133).
- [SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. viii–x, 10, 13, 45, 59, 67, 71, 73, 81, 97–99, 115, 119, 122, 130–133, 141, 147, 163, 170).
- [STY01] Tomas Sander, Amnon Ta-Shma, and Moti Yung. *Blind, Auditable Membership Proofs*. In: *FC 2000*. Ed. by Yair Frankel. Vol. 1962. LNCS. Springer, Heidelberg, Feb. 2001, pp. 53–71 (cit. on p. 122).
- [TT05] Roberto Tamassia and Nikos Triandopoulos. *Computational Bounds on Hierarchical Data Processing with Applications to Information Security*. In: *ICALP 2005*. Ed. by Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung. Vol. 3580. LNCS. Springer, Heidelberg, July 2005, pp. 153–165 (cit. on pp. 120, 122, 129).

Invincibility lies in the defence; the possibility of victory in the attack.

*The Art of War* – SUN TZU

# Leakage Abuse Attacks and How to Thwart Them

# 7

**L**EAKAGE IS FUNDAMENTAL FOR SEARCHABLE ENCRYPTION, as we saw in previous chapters. Some minimal leakage is required to achieve good performance. From the security definitions and the schemes' security proof, we can quickly ensure that the server cannot learn more information than what the leakage function prescribes. But what does the leakage means in practice? Up to what extent the allowed leakage is not already too much?

One step towards the understanding of leakage is the recent development of *leakage abuse attacks* that aim at decrypting queries and/or the database using only the leaking information [IKK12; CGPR15; KKNO16]. None of these attacks invalidate in any way the security proofs of the schemes they target: they only use the leakage to break the confidentiality of the database or of the queries. Yet, we will see that they all fall outside of the security model. Indeed, none of the possible adversarial knowledge described in [CGPR15] is captured by the security definition.

In this chapter, following our work with Fouque [BF17], we will try to understand the discrepancy between the security proofs and the leakage abuse attacks, we will give new security definitions capturing the attacks, and finally provide methods to assess the security of existing constructions, and use them to construct schemes provably secure against such attacks.

## Contents

---

<b>7.1</b>	<b>Leakage Abuse Attacks and their Origin</b>	<b>152</b>
<b>7.2</b>	<b>Fixing the Security Definition</b>	<b>153</b>
7.2.1	Constraints	153
7.2.2	Constrained Security	155
7.2.3	Examples of Constraints	156
7.2.4	Devising New Leakage Abuse Attacks Using Constraints	157
7.2.5	Extending the Security Definition	157
<b>7.3</b>	<b>Keywords Clustering</b>	<b>160</b>
7.3.1	Regrouping Keywords with Equal Leakage	160
7.3.2	Applications to Common Leakage Profiles and Constraints	161
<b>7.4</b>	<b>Application to Database Padding with Best Possible Security</b>	<b>163</b>
7.4.1	Using Frequencies Instead of Counts	163
7.4.2	How to Pad	164
7.4.3	Constructing Frequency-based Clusters	165
7.4.4	Integration to Existing Schemes	170
<b>7.5</b>	<b>Experiments</b>	<b>170</b>
7.5.1	The Performance of the Clustering Algorithm	170
7.5.2	Influence of Secure Padding on the Count Attack	171

---

## 7.1 Leakage Abuse Attacks and their Origin

The terminology ‘*inference attack*’ was introduced by Islam *et al.* in [IKK12], who used co-occurrence information on an encrypted database. Their attack was improved by Cash *et al.* [CGPR15], who used the term ‘*leakage abuse attack*’ to describe attacks that only use the leakage of a scheme to break its security, rather than exploiting some particular weakness in the construction.

Both of these attacks suppose knowledge (total, partial or distributional) of the database by the adversary. In [IKK12], he knows the distribution  $\mathcal{D}$  of the co-occurrence matrix of the targeted set of keywords. From the observation of the document access pattern (the list of results), the adversary builds a co-occurrence matrix that follows the distribution  $\mathcal{D}$ , up to rows and columns permutation. Using simulated annealing, he will find this permutation, which will be the match between queries and keywords.

The count attack of [CGPR15] also uses a co-occurrence matrix, but uses it only to leverage some prior knowledge issued from the uniqueness of the number of results for some keywords: there are some keywords  $w$  such that no other keyword  $w'$  match the exact same number of documents. Hence, for the full or partial knowledge of the database, the adversary is able to decrypt the queries keyword.

On the other side, the attack of Kellaris *et al.* [KKNO16] targets range queries, and only supposes that the queries are performed uniformly at random to recover the entire dataset, using only the number of results for each query. In particular, this attack is successful even against ORAM-based constructions: even though they only leak the result count, this is sufficient to break the dataset’s secrecy.

Note that all these attacks are passive and hence, non-adaptive: the adversary does not choose either the database, or the queries. However, adaptive attacks do exist: some very efficient files injection attacks presented by Zhang *et al.* [ZKP16] are adaptive and fall in the category of leakage abuse attacks. In order to decrypt a previous search query, the adversary uses the update leakage of the scheme and adaptively inserts a sequence of well crafted documents in the database. Yet, as we saw in Chapter 4 these adaptive attacks can be circumvented using *forward-secure* searchable encryption schemes, and to our knowledge, except these ones, all the existing leakage abuse attacks are not adaptive.

**Why does leakage abuse work?** A very important question that consequently arises is the following. Why schemes that are proven secure in the indistinguishability-based security model can be broken only using the leakage, while the definition states that the adversary should not be able to distinguish two executions of the SE scheme with the same leakage?

This paradox is particularly apparent for all the leakage targeted by the previously mentioned works, as, for these leakage profiles, the very important observation of Curtmola *et al.* [CGKO06, Section 4.2] stands:

*Note that the existence of a second history with the same trace is a necessary assumption, otherwise the trace would immediately leak all information about the history.*

Namely, it is fairly easy, given a database and queries list, to construct an *other* database and queries list with the exact same leakage, for all the common leakage used in searchable encryption (e.g. one could permute the database’s keywords): histories are non-singular.

Yet, attacks like the one of Islam *et al.* [IKK12] or the count attack of [CGPR15] suppose some server knowledge of the database. This knowledge *pins* the database in the security proof: for the proof to be useful in this setting, one needs to find two different lists of queries generating the exact

same leakage with the same (public) database. And the whole point of leakage abuse attacks is that this is impossible: knowing the database, the queries' list is uniquely defined by the leakage. For example, once the database is committed to, one cannot permute the keywords anymore to construct a different history with the same leakage. Also, the frequency of words in the English language is fixed, so if the adversary knows that a dataset stores some English-written documents, any syntactic permutation of the keywords would not hide the real keywords.

A similar point can be made with the attacks in [KKNO16]: queries are known to be uniform, and many of them are performed. And for the active attacks of Zhang *et al.* [ZKP16] too, but instead of controlling the database, the attacker controls some update queries as he injects documents he has purposely built.

In all of these examples, the adversary can decrypt queries and/or the database because they are unique given the constraints (fixed database, knowledge of the queries distribution, knowledge or control of some updates, ...) and the leakage.

Another way to see this issue with security definitions for SE is the following: the existing definitions protect the database and the queries as a whole, but once one part gets leaked (e.g. the database in [CGPR15]) these definitions are of no use anymore. If one only wants to protect the queries, it might be more suitable to use private information retrieval (PIR) definitions [KO97].

**On the meaningfulness of security definitions.** A parallel can be made with the CPA security definition for encryption: the definition states that the adversary has to give two messages of equal length to the challenger. If the message space contains a message with a unique length, this message is not protected by the security definition. If every element of the message space has a unique length, any scheme (e.g. a scheme whose encryption function is the identity) can be shown CPA-secure although it is trivially insecure: it will be impossible for an adversary to find a pair of same length messages on which to be challenged. In this specific setting, the CPA security definition is void, as is the indistinguishability based security definition for SE with prior knowledge.

## 7.2 Fixing the Security Definition

We saw in the previous section that searchable encryption fails against leakage abuse attacks because there is uniqueness of histories given some constraints (the leakage plus other external constraints such as the distribution of queries or the publicity of the dataset). To fix this problem, a searchable encryption scheme's leakage function must be so that, given a constraint, histories are no longer uniquely defined by the leakage function.

In this section, we propose new tools and definitions to capture leakage abuse attacks. Also, to avoid verbosity, in this chapter, we unify search and updates queries. Both will be denoted with the letter  $q$  and the execution of the query will be  $\text{Query}(q)$ , the Query function making the dispatch between searches and updates. We proceed similarly for  $\mathcal{L}^{\text{Query}}$ .

### 7.2.1 Constraints

Defining constraints is a way to formalize that the adversary knows some information about the history. In particular, we must be able to tell if a history *conforms* to the information known by the adversary. We could ask the adversary to represent this information as the known database plus the list of queries he knows, but that would be overly restrictive: we would not be able to represent partial knowledge.

A more general way to represent this knowledge is by using constraints defined by a predicate over histories: the history  $H$  satisfies the constraint  $C$  if and only if  $C(H) = \text{true}$ . However, we need an adaptive way to define constraints: the adversary may want to insert a document depending on the transcript of previous queries. This is what Definition 7.1 captures.

**Definition 7.1** (Constraint). *A constraint  $C = (C_0, C_1, \dots, C_n)$  (with  $n = \text{poly}(\lambda)$ ) over a set of databases  $\mathcal{DB}$  and set of queries  $\mathcal{Q}$  is a sequence of algorithms such that, for  $\text{DB} \in \mathcal{DB}$ ,*

$$C_0(\text{DB}) = (\text{flag}_0, st_0)$$

where  $\text{flag}_0$  is true or false and  $st_0$  captures  $C_0$ 's state, and for  $q \in \mathcal{Q}$ ,

$$C_i(q, \text{flag}_{i-1}, st_{i-1}) = (\text{flag}_i, st_i).$$

The constraint is consistent if  $C_i(\cdot, \text{false}, \cdot) = (\text{false}, \cdot)$  (once the constraint evaluates to false, it remains false).

For a history  $H = (\text{DB}, q_1, \dots, q_n)$ , we note  $C(H)$  the evaluation of

$$C(H) := C_n(q_n, C_{n-1}(q_{n-1}, C_{n-2}(\dots, C_0(\text{DB}))))).$$

If  $C(H) = \text{true}$ , we say that  $H$  satisfies  $C$ . A constraint  $C$  is valid if there exists two different efficiently constructible histories  $H$  and  $H'$  satisfying  $C$ .

The validity of the constraint makes sure that the adversary does not know everything about the history. Note that this definition of constraints does not extend to distributional knowledge, and in Section 7.2.5.2, we study a security definition supporting prior distributional knowledge by the adversary. In the following, and except for Section 7.2.5.2, we will restrict ourselves to deterministic knowledge by the server. Also, to simplify the notations, we will omit passing the states  $st_i$ . In this chapter, we will only consider valid constraints.

We also want to formalize the fact that some elements of the history are completely unknown to the adversary, i.e. that their are left *free* from constraint.

**Definition 7.2** (Free history component). *Let  $C$  be a constraint. We say that  $C$  lets the database free if, for every history  $H = (\text{DB}, q_1, \dots, q_n)$  satisfying  $C$ , for every  $\text{DB}' \in \mathcal{DB}$ ,  $H' = (\text{DB}', q_1, \dots, q_n)$  also satisfies  $C$ .*

*We say the  $C$  lets the  $i$ -th query free if for every history  $H = (\text{DB}, q_1, \dots, q_n)$  satisfying  $C$ , for every search (resp. update) query  $q$  if  $q_i$  is a search (resp. update) query,  $H' = (\text{DB}', q_1, \dots, q_{i-1}, q, q_{i+1}, \dots, q_n)$  also satisfies  $C$ .*

Finally, another very important notion is the one of *acceptable* constraint, that will help us to give non-void security definition: as explained in Section 7.1, given a constraint  $C$  and a leakage function  $\mathcal{L}$ , for every history  $H$  we want to be able to find a different history satisfying  $C$  with the same leakage.

**Definition 7.3** (Acceptable constraint). *A constraint  $C$  is  $\mathcal{L}$ -acceptable for some leakage  $\mathcal{L}$  if, for every efficiently computable history  $H$  satisfying  $C$ , there exists an efficiently computable  $H' \neq H$  satisfying  $C$  such that  $\mathcal{L}(H) = \mathcal{L}(H')$ .*

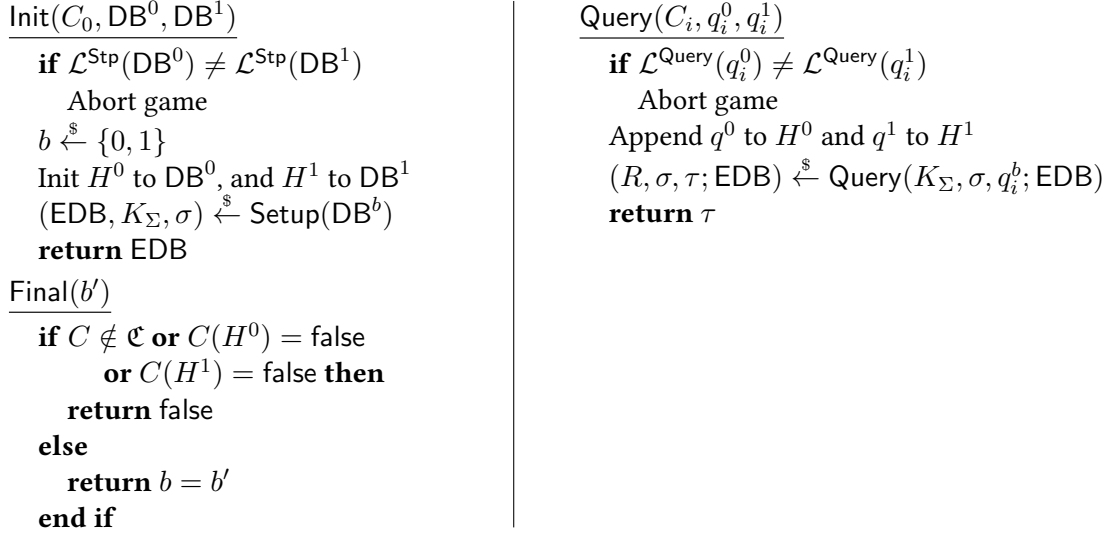
*A set of constraints  $\mathcal{C}$  is said to be  $\mathcal{L}$ -acceptable if all its elements are  $\mathcal{L}$ -acceptable.*

Section 7.2.3 will give examples of constraints. We first explain how we will formally use this new tool.

### 7.2.2 Constrained Security

Now that we have formally defined what are the constraints, we can use them to give a new flavor of security for history satisfying these constraints.

**Definition 7.4** (Constrained adaptive indistinguishability). *Let  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$  be an SE scheme,  $\lambda$  the security parameter, and  $A$  a stateful algorithm. Let  $\mathcal{L}$  be a leakage function and  $\mathfrak{C}$  be a set of  $\mathcal{L}$ -acceptable constraints. We can define the notion of constrained adaptive indistinguishability using the  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathfrak{C}}$  game described in Figure 7.1.*



**Figure 7.1** –  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathfrak{C}}$ : Constrained indistinguishability game for the SSE scheme  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ , with the leakage function  $\mathcal{L} = (\mathcal{L}^{\text{Stp}}, \mathcal{L}^{\text{Srch}}, \mathcal{L}^{\text{Updt}})$ , and family of constraints  $\mathfrak{C}$ .

$\text{SSEIND}_{\Sigma, \mathcal{L}, \mathfrak{C}}$  is almost identical to  $\text{SSEIND}_{\Sigma, \mathcal{L}}$  (Figure 3.2), with the restrictions that, for  $H^0 = (\text{DB}^0, q_1^0, \dots, q_n^0)$  and  $H^1 = (\text{DB}^1, q_1^1, \dots, q_n^1)$ ,

- $C \in \mathfrak{C}$ ,  $C(H^0) = \text{true}$ , and  $C(H^1) = \text{true}$  ;
- $\mathcal{L}(H^0) = \mathcal{L}(H^1)$ .

We say that  $\Sigma$  is  $(\mathcal{L}, \mathfrak{C})$ -constrained-adaptively-indistinguishable if for any polynomial-time adversary  $A$ ,

$$\text{Adv}_{\Sigma, \mathcal{L}, \mathfrak{C}, A}^{\text{SSE-ind}}(\lambda) = \left| \frac{1}{2} - \mathbb{P}[\text{SSEIND}_{\Sigma, \mathcal{L}, \mathfrak{C}}^A(\lambda) = 1] \right| \leq \text{negl}(\lambda).$$

Definition 7.4 is similar to the original SE security definition (Definition 3.3), with the main difference being the introduction of the condition that both histories must satisfy the constraint.

Note that a weaker, non-adaptive security notion can be easily derived from Definition 7.4 by making the adversary output the whole constraint and histories at once, at the beginning of the game.

If the leakage function is probabilistic, we replace the last restriction by the condition that the distribution of  $\mathcal{L}(H^0)$  and  $\mathcal{L}(H^1)$  must be indistinguishable, as defined in Section 3.1.3.2 for the regular security definition.

We underline again that the constraint can be seen as some information the server knows about the histories: the histories both have to satisfy the same constraint.

The fact that the constraints are acceptable implies that the definition is not void. Also, we can prove the following theorem, stating that we only have to prove (resp. give counter-examples of) the acceptability of some constraints given some common leakage function  $\mathcal{L}$  to show the security (resp. insecurity) of existing schemes.

**Theorem 7.1.** *Let  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$  be an SE scheme, and  $\mathcal{C}$  a set of constraints. If  $\Sigma$  is  $\mathcal{L}$ -adaptive-indistinguishability secure, and  $\mathcal{C}$  is  $\mathcal{L}$ -acceptable, then  $\Sigma$  is  $(\mathcal{L}, \mathcal{C})$ -constrained-adaptive-indistinguishability secure.*

*Proof.* Suppose  $\mathcal{C}$  is  $\mathcal{L}$ -acceptable. Then, for any satisfying pair  $(H^0, H^1)$  of histories such that  $\mathcal{L}(H^0) = \mathcal{L}(H^1)$ , the views of the adversary will be indistinguishable, from the  $\mathcal{L}$ -adaptive-indistinguishability.  $\square$

### 7.2.3 Examples of Constraints

Using these constraints, it is easy to model the fact that the adversary knows some information about the database or about some queries. This section gives example of constraints for existing leakage abuse attacks.

**Prior knowledge of the database.** Let us consider the setting of the count attack of Cash *et al.* [CGPR15]: the adversary knows the database DB and uses the leakage of the search queries to decrypt them. In the security definition, we want to capture that the adversary knows DB.

To do so, we will use the predicate  $C^{\text{DB}}$  that returns true if and only if the database of the input history is DB. Used in the security definition, this predicate will ensure that the both challenge histories' database is DB, and that all the queries are left free. [CGPR15, Section 4.2] shows that, for the leakage function  $\mathcal{L} = L1$  (the repetition of queries, the number of documents matching each queries and, for every pair of queries, the documents in common),  $C^{\text{DB}}$  is not  $L1$ -acceptable: many keywords have a unique number of matching documents, and as the adversary knows the database, queries on these keywords can be decrypted just from the results count, and all the others queries from co-occurrence information.

More generally, we model the fact that the adversary knows the database by considering the set  $\mathcal{C}^{\mathcal{DB}} = \{C^{\text{DB}}, \text{DB} \in \mathcal{DB}\}$  where  $\mathcal{DB}$  is the set of polynomially computable databases.

**Known documents subset.** We similarly define the partial knowledge of the dataset: if the adversary knows that the database contains documents  $D_1, \dots, D_\ell$ , we will use the constraint  $C^{D_1, \dots, D_\ell}$  that returns true if and only if DB contains  $D_i$  for all  $i$ . Cash *et al.* also showed that for the leakage function  $\mathcal{L} = L3$  (which leaks the pattern of keyword occurrences in documents – but not the occurrence count),  $C^{D_1, \dots, D_\ell}$  is not  $L3$  acceptable [CGPR15, Section 5.1].

**File injection attacks.** File injection attacks (*cf.* Section 4.1 and [ZKP16]) are also captured by this formalization. Say the attacker inserts  $\ell$  documents  $D_1, \dots, D_\ell$ ,  $D_j$  being inserted during the  $i_j$ -th query. We construct  $C$  so that  $C_0(\cdot)$  always outputs true (the adversary does not know the database at the beginning),  $C_{i_j}(\text{flag}, q)$  outputs true if and only if flag is true and  $q$  is the query inserting  $D_j$  in the database (the  $i_j$ -th query is forced to be the insertion of  $D_{i_j}$ ), and  $C_i(\text{flag}, q)$  outputs flag when  $i \notin \{i_1, \dots, i_\ell\}$  (for all the other queries, there is no constraint).



This constraint can be used both for the non-adaptive and the adaptive attacks of Zhang *et al.* For the non-adaptive attack, the adversary will choose the  $D_j$  so that he will be able to run a binary search for all the subsequent search queries. For the adaptive attack, to break the privacy of a previous search query, the adversary will choose the successive documents to be inserted using the update leakage of the previously insertion query. We refer to [ZKP16] and Section 4.1 for further details on these attacks.

As before, we can generalize all file injection attacks by considering the constraints for all the polynomially constructible lists of pairs  $\{(D_j, i_j)\}_{1 \leq j \leq \ell}$ . Hence, [ZKP16] shows that this set of constraints is not  $\mathcal{L}$ -acceptable when  $\mathcal{L}$  leaks the file-access pattern (which is the case for all the existing non-ORAM-based SE schemes).

### 7.2.4 Devising New Leakage Abuse Attacks Using Constraints

For now, we have used constraints as a way to formalize the security of schemes against leakage abuse attacks. But we can also use them as a way to construct new attacks.

Indeed, with leakage abuse attacks, we suppose that the targeted scheme is  $\mathcal{L}$ -indistinguishable for some leakage function  $\mathcal{L}$ , and breaking the  $(\mathcal{L}, \mathfrak{C})$ -constrained-indistinguishability for some set of constraints  $\mathfrak{C}$  implies that  $\mathfrak{C}$  is not  $\mathcal{L}$ -acceptable. To mount a new attack, one could then just check if there exists an history satisfying  $C \in \mathfrak{C}$  such that no other history satisfying  $C$  has the same leakage. Even though this step can be (very) fastidious by hand, it can be automated using constraint programming. However, we would have to make sure that the adversarial knowledge induced by the constraint is realistic and reasonable (e.g. that there is not too much prior knowledge).

### 7.2.5 Extending the Security Definition

The new constrained security definition, despite being satisfactory because it solves the definitional issue we had with leakage abuse attacks, still is not enough in terms of real-world security.

#### 7.2.5.1 Extension to More Than Two Histories

Indeed, it ensures that the adversary will not be able to distinguish between two histories. Yet, when proposed  $k$  different histories satisfying the same constraint and sharing the same leakage, he could be able to easily discard one of these. Said otherwise, Definition 7.4 guarantees us that there is some uncertainty for the adversary, but not how much: the security definition only guarantees one bit of security.

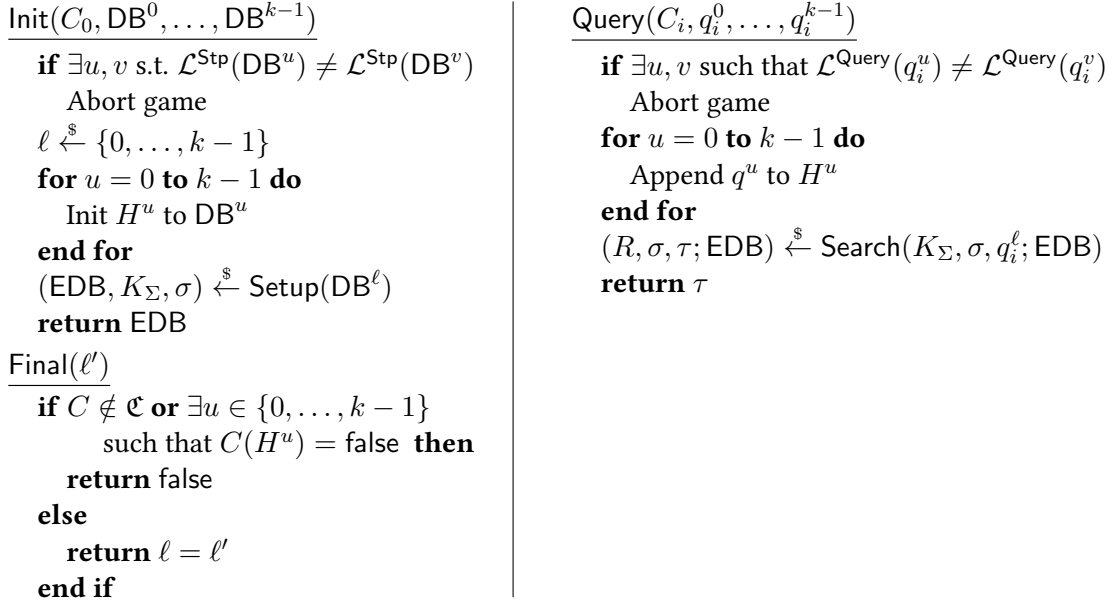
However, we can extend it by modifying the  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathfrak{C}}^A(\lambda)$  game: instead of adaptively outputting a pair of histories,  $A$  could output  $k$  of them, and the challenger randomly picks the one to be guessed. For this definition to make sense, we will have to make sure that  $k$  constrained histories can actually be found, and we also have to extend the definition of acceptable constraints.

**Definition 7.5** (Extended acceptable constraint). *A constraint  $C$  is  $(\mathcal{L}, k)$ -acceptable for some leakage  $\mathcal{L}$  and integer  $k > 1$  if, for every efficiently computable history  $H^0$  satisfying  $C$  ( $C(H^0) = \text{true}$ ), there exists  $k - 1$  efficiently computable  $\{H^i\}_{1 \leq i \leq k-1}$  such that  $H^i \neq H^j$  for  $i \neq j$ , that are all satisfying  $C$ , and  $\mathcal{L}(H^0) = \dots = \mathcal{L}(H^{k-1})$*

*A set of constraints  $\mathfrak{C}$  is said to be  $(\mathcal{L}, k)$ -acceptable if and only if all its elements are  $(\mathcal{L}, k)$ -acceptable.*

**Definition 7.6** (Extended constrained adaptive indistinguishability). *Let  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$  be an SE scheme,  $\lambda$  the security parameter, and  $A$  a stateful algorithm. Let  $\mathfrak{C}$  be a set of*

$(\mathcal{L}, k)$ -acceptable constraints. We can define the notion of constrained adaptive indistinguishability using the  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{C}, k}$  game described in Figure 7.2.



**Figure 7.2** –  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{C}, k}$ : Extended constrained indistinguishability game for the SSE scheme  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ , with the leakage function  $\mathcal{L} = (\mathcal{L}^{\text{Stp}}, \mathcal{L}^{\text{Srch}}, \mathcal{L}^{\text{Updt}})$ , family of constraints  $\mathcal{C}$ , and with  $k$  possible histories.

We say that  $\Sigma$  is  $(\mathcal{L}, \mathcal{C}, k)$ -constrained-adaptively-indistinguishable if for any polynomial-time adversary  $A$ ,

$$\text{Adv}_{\Sigma, \mathcal{L}, \mathcal{C}, k, A}^{\text{SSE-ind}}(\lambda) = \left| \frac{1}{k} - \mathbb{P}[\text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{C}, k}^A(\lambda)] \right| \leq \text{negl}(\lambda).$$

An  $(\mathcal{L}, \mathcal{C}, k)$ -constrained-adaptively-indistinguishable scheme offers at least  $\log k$  bits of security. Extended constrained indistinguishability is implied by regular indistinguishability and extended acceptability of constraints, as stated in Theorem 7.2.

**Theorem 7.2.** *Let  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$  be an SE scheme, and  $\mathcal{C}$  a set of constraints. If  $\Sigma$  is  $\mathcal{L}$ -adaptive-indistinguishability secure, and  $\mathcal{C}$  is  $(\mathcal{L}, k)$ -acceptable, then  $\Sigma$  is  $(\mathcal{L}, \mathcal{C}, k)$ -constrained-adaptive-indistinguishability secure.*

*Proof.* From Theorem 7.1, we know that  $\Sigma$  is  $(\mathcal{L}, \mathcal{C})$ -constrained-adaptive-indistinguishability secure. So we must show that  $(\mathcal{L}, \mathcal{C})$ -constrained-adaptive-indistinguishability implies  $(\mathcal{L}, \mathcal{C}, k)$ -constrained-adaptive-indistinguishability. Note that, as  $\mathcal{C}$  is  $(\mathcal{L}, k)$ -acceptable, the  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{C}, k}^A$  game is not void.

Let  $A$  be an adversary in the  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{C}, k}^A$  game. We construct an adversary  $\mathcal{B}$  against the  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{C}}^A$  game in the following way.

$\mathcal{B}$  starts by randomly picking two integers  $k_0, k_1 \in \{0, \dots, k-1\}$ . Then,  $\mathcal{B}$  starts  $A$  and receives  $k$  databases  $(\text{DB}^0, \dots, \text{DB}^{k-1})$ . Upon giving the pair  $(\text{DB}^{k_0}, \text{DB}^{k_1})$  to the challenger,  $\mathcal{B}$  receives the challenge  $\text{EDB}^*$  which he forwards to  $A$ . Then  $A$  repeatedly outputs  $k$  queries  $(q_i^0, \dots, q_i^{k-1})$ ,  $\mathcal{B}$  outputs  $(q_i^{k_0}, q_i^{k_1})$  to the game, receives back the transcript  $\tau_i^*$  and forwards it to  $A$ . Eventually  $A$  outputs an integer  $k'$ . If  $k' = k_0$ ,  $\mathcal{B}$  output  $b' = 0$ , if  $k' = k_1$ ,  $\mathcal{B}$  outputs  $b' = 1$ , and otherwise outputs 0 with probability 1/2 and 1 with probability 1/2.

We know have to evaluate  $\mathbb{P}[b = b']$ , where  $b$  is the random bit picked by the challenger.

$$\begin{aligned}\mathbb{P}[b = b'] &= \mathbb{P}[b = b' \cap k' \in \{k_0, k_1\}] + \mathbb{P}[b = b' | k' \notin \{k_0, k_1\}] \cdot \mathbb{P}[k' \notin \{k_0, k_1\}] \\ &= \mathbb{P}[A \text{ wins the } \text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{E}, k}^A \text{ game}] + \frac{1}{2}(1 - \mathbb{P}[k' \in \{k_0, k_1\}])\end{aligned}$$

We can also evaluate  $\mathbb{P}[k' \in \{k_0, k_1\}]$ :

$$\begin{aligned}\mathbb{P}[k' \in \{k_0, k_1\}] &= \mathbb{P}[k' = k_0] + \mathbb{P}[k' = k_1] \\ &= \frac{1}{2}(\mathbb{P}[k' = k_b | b = 0] + \mathbb{P}[k' = k_1]) + \frac{1}{2}(\mathbb{P}[k' = k_0] + \mathbb{P}[k' = k_b | b = 1])\end{aligned}$$

As  $k_0$  and  $k_1$  are uniformly picked in  $\{0, \dots, k-1\}$ , and as  $\mathbb{P}[k' = k_b]$  is the probability that  $A$  wins the 1-out-of- $k$  indistinguishability game, we have

$$\mathbb{P}[k' \in \{k_0, k_1\}] = \mathbb{P}[A \text{ wins the } \text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{E}, k}^A \text{ game}] + \frac{1}{k}$$

Finally, we can conclude that

$$\text{Adv}_{\Sigma, \mathcal{L}, \mathcal{E}, A}^{\text{SSE-ind}}(\lambda) = \frac{1}{2} \text{Adv}_{\Sigma, \mathcal{L}, \mathcal{E}, k, A}^{\text{SSE-ind}}(\lambda).$$

□

### 7.2.5.2 Extension to Distributional Knowledge

We saw in Section 7.2.1 that the constraint-based definition did not capture distributional knowledge of the adversary on the database or on the queries. Here we propose a variation of the constrained adaptive indistinguishability security definition that will capture this kind of prior knowledge.

The idea is that the adversary, instead of a history and a constraint, will give to the challenger a pair of history distributions that will be used to sample the challenges. Before giving the actual security definition, we have to define the notion of *acceptable set of distributions*, similarly to the way we defined acceptable constraints in Section 7.2.1. In the following, for a distribution  $\mathcal{D}$  of histories, we define  $\mathcal{L}(\mathcal{D})$  as the distribution of  $\{L(H) \text{ s.t. } H \xleftarrow{\$} \mathcal{D}\}$

**Definition 7.7** (Acceptable set of distribution). *A set  $\mathcal{D}$  of history distributions is  $\mathcal{L}$ -acceptable for some leakage  $\mathcal{L}$  if, for every efficiently computable distribution  $\mathcal{D} \in \mathcal{D}$ , there exists an efficiently computable  $\mathcal{D}' \in \mathcal{D}$  different from  $\mathcal{D}$  such that  $\mathcal{L}(\mathcal{D})$  and  $\mathcal{L}(\mathcal{D}')$  are indistinguishable.*

**Definition 7.8** (Distribution indistinguishability for SE). *Let  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$  be an SE scheme,  $\lambda$  the security parameter, and  $A$  a stateful algorithm. Let  $\mathcal{D}$  be a set of  $\mathcal{L}$ -acceptable distribution. We can define the notion of distribution adaptive indistinguishability using the  $\text{SSEIND}_{\mathcal{L}, \mathcal{D}}^A(\lambda)$  game define in Figure 7.3. In the game, we decompose a history distribution  $\mathcal{D}$  as a sequence of database and query distribution  $(\mathcal{D}, \mathcal{Q}_1, \dots, \mathcal{Q}_n)$*

*We say that  $\Sigma$  is  $(\mathcal{L}, \mathcal{D})$ -distribution-adaptively-indistinguishable if for any polynomial-time adversary  $A$ ,*

$$\text{Adv}_{\Sigma, \mathcal{L}, \mathcal{D}, A}^{\text{SSE-ind}}(\lambda) = \left| \mathbb{P}[\text{SSEIND}_{\mathcal{L}, \mathcal{D}}^A(\lambda) = 1] - \frac{1}{2} \right| \leq \text{negl}(\lambda).$$

<pre> Init(<math>\mathcal{DB}^0, \mathcal{DB}^1</math>)   if <math>\mathcal{L}^{\text{Stp}}(\mathcal{DB}) \not\approx \mathcal{L}^{\text{Stp}}(\mathcal{DB})</math>     Abort game   <math>b \xleftarrow{\\$} \{0, 1\}</math>   <math>\text{DB} \xleftarrow{\\$} \mathcal{DB}^b</math>   <math>(\text{EDB}, K_\Sigma, \sigma) \xleftarrow{\\$} \text{Setup}(\text{DB})</math>   return EDB Final(<math>b'</math>)   if <math>\mathcal{D}^0 \in \mathcal{D}</math> and <math>\mathcal{D}^1 \in \mathcal{D}</math>     return <math>b = b'</math>   return 0 </pre>	<pre> Query(<math>Q_i^0, Q_i^1</math>)   if <math>\mathcal{L}^{\text{Query}}(Q_i^0) \not\approx \mathcal{L}^{\text{Query}}(Q_i^1)</math>     Abort game   <math>q \xleftarrow{\\$} Q^b</math>   <math>(R, \sigma, \tau; \text{EDB}) \xleftarrow{\\$} \text{Search}(K_\Sigma, \sigma, q; \text{EDB})</math>   return <math>\tau</math> </pre>
---	--

**Figure 7.3** –  $\text{SSEIND}_{\Sigma, \mathcal{L}, \mathcal{D}}$ : Distribution indistinguishability game for the SSE scheme  $\Sigma = (\text{Setup}, \text{Search}, \text{Update})$ , with the leakage function  $\mathcal{L} = (\mathcal{L}^{\text{Stp}}, \mathcal{L}^{\text{Srch}}, \mathcal{L}^{\text{Updt}})$ , and family of history distribution  $\mathcal{D}$ .

If we look at Kellaris *et al.*'s paper [KKNO16], we can see that our security definition actually capture their adversarial setting for their attack on encrypted databases supporting range queries. Indeed the set of distribution to be considered is the set  $\mathcal{D}^{\text{DB}} = \{\mathcal{D}^{\text{DB}} | \text{DB} \in \mathcal{DB}\}$  of distribution  $\mathcal{D}^{\text{DB}}$  such that the database distribution is reduced to 'deterministically' output only one element DB, and the queries distribution is uniform over all the range queries.

Kellaris *et al.* show that, for unbounded histories,  $\mathcal{D}^{\text{DB}}$  is not  $\mathcal{L}$ -acceptable, where  $\mathcal{L}$  is the function leaking the number of results of a query [KKNO16, Section 4].

## 7.3 Keywords Clustering

Section 7.2 gives us the formal tools to assess the security of a SE construction. Yet these tools are not constructive: for a given constraint and history, it is hard to tell how many other histories satisfying the constraint have the same leakage, or even if one exists.

In this section, we propose an easy way to evaluate the security of leakage functions for a usual class of constraints (partial or complete knowledge of the database): this approach will allow for queries' privacy protection.

### 7.3.1 Regrouping Keywords with Equal Leakage

We will suppose as, a starting point, that the leakage function only depends on the query itself and on the state of the database:  $\mathcal{L}(q)$  can be written as a stateless function  $f_{\mathcal{L}}$  of  $q$  and DB.

We make the following very simple observation: let  $C$  be a constraint,  $H = (\text{DB}, q_1, \dots, q_n)$  an history satisfying  $C$ , and  $q, q'$  be two queries such that  $\tilde{H} = H || q = (\text{DB}, q_1, \dots, q_n, q)$  and  $\tilde{H}' = H || q' = (\text{DB}, q_1, \dots, q_n, q')$  are both satisfying  $C$ . Then, if  $f_{\mathcal{L}}(\text{DB}, q) = f_{\mathcal{L}}(\text{DB}, q')$ ,  $\tilde{H}$  and  $\tilde{H}'$  are two histories with the same leakage satisfying  $C$ . This observation can be iterated to easily create histories satisfying the constraint and with the same leakage, showing the acceptability of the said constraint.

More generally, a clustering  $\Gamma = \{G_1, \dots, G_m\}$  of queries induced by the leakage  $\mathcal{L}$  after history  $H$  is a partition of the queries set  $\mathcal{Q}$  for which, in every subset, queries share the same leakage after

running the history  $H$ :

$$\begin{aligned} \bigcup_{i=1}^m G_i &= \mathcal{Q} \\ \forall i \neq j, G_i \cap G_j &= \emptyset \\ \text{and } \forall q, q' \in G_i, \mathcal{L}(H, q) &= \mathcal{L}(H, q'), \end{aligned}$$

where  $\mathcal{L}(H, q)$  is the output of  $\mathcal{L}(q)$  after having been run on each element of  $H$ . In the following, we denote by  $\Gamma_{\mathcal{L}}(H)$  the clustering induced by  $\mathcal{L}$  after  $H$ , *i.e.* the clustering for which it is impossible to *merge* clusters with the same leakage. Formally, for  $\Gamma_{\mathcal{L}}(H) = \{G_1, \dots, G_m\}$ , we have

$$\forall i \neq j, \forall q \in G_i, \forall q' \in G_j, \mathcal{L}(H, q) \neq \mathcal{L}(H, q').$$

Also, we denote by  $\Gamma_{\mathcal{L}, C}(H)$  the clustering of the subset  $\mathcal{Q}_C(H)$  of queries  $q$  such that  $C(H||q) = \text{true}$ . We can easily see that, in the specific case studied before, where  $\mathcal{L}(q)$  is a function of DB and  $q$  only,  $\Gamma_{\mathcal{L}}$  only depends on DB and, in the static case, not on previous queries.

We want that every cluster of  $\Gamma_{\mathcal{L}, C}(H)$  contains at least two elements. If this is not the case, one is able to construct an history  $H$  satisfying  $C$  without having any different history  $H'$  with the same leakage profile, also satisfying  $C$ :  $C$  will not be  $\mathcal{L}$ -acceptable, and this may lead to a new leakage abuse attack. Yet, this only makes sense if  $|\mathcal{Q}_C(H)| > 1$  (there is more than one satisfying query).

In the opposite, we can show that when there is strictly more than one element in each cluster of the  $\mathcal{L}$ -induced clustering applied on every history satisfying  $C$ ,  $C$  will be  $\mathcal{L}$ -acceptable, as formalized in Proposition 7.3.

**Proposition 7.3.** *Let  $C$  be a constraint, and  $\mathcal{L}$  a leakage function. If for every history  $H$  satisfying  $C$ , the clustering  $\Gamma_{\mathcal{L}, C}(H) = \{G_1, \dots, G_m\}$  is such that  $|G_i| \geq k$  for all  $i$ , then  $C$  is  $(\mathcal{L}, k)$ -acceptable.*

Unfortunately, the condition that  $|G_i| \geq k$  is very strong: constraints fixing a particular query will never verify it. On the other side, it looks very hard to give a better result. Take for example a static scheme whose leakage function  $\mathcal{L}$  gives away the search pattern, *i.e.* for a search query  $q$  after the queries  $(q_1, \dots, q_n)$ , the set  $\text{sp}(q) = \{i | q_i = q\}$ , and consider the constraint  $C(H)$  which return true if and only if  $\text{DB} = \overline{\text{DB}}$  and  $q_2 = q$  (the database and second query are fixed). Then  $C^{2, q}$  will not be  $\mathcal{L}$ -acceptable:  $H = (\overline{\text{DB}}, q, q)$  has no matching history satisfying  $C$  with the same leakage.

As the search pattern is leaked for almost all practical SE schemes, we can see that generic constrained security is very hard to achieve. Hence, in the following, we will restrict ourselves to common constraints, *i.e.* common adversarial prior knowledge in leakage abuse attacks.

### 7.3.2 Applications to Common Leakage Profiles and Constraints

In this section, we will see how to use the clustering approach to assess the security of single-keyword SE schemes with simple, yet common leakage, against existing attacks. Let us first focus on static schemes.

**Result length leakage.** All (reasonably efficient) schemes leak at least the number of matches of a search query, and even this leakage can break the queries' privacy, as shown by Cash *et al.* [CGPR15] (63% of the 500 most common words of the Enron database have a unique result count). In particular, this implies that ORAM-based schemes [GMP16], whose leakage  $\mathcal{L}_{len}$  is limited to the size of the

database and to the number of queries' matches, are not resilient to leakage abuse attacks when the adversary knows the database:  $\mathfrak{C}^{\mathcal{DB}}$  is not  $\mathcal{L}_{len}$ -acceptable.

Now, suppose that we add to a scheme with  $\mathcal{L}_{len}$  leakage a padding mechanism such that, for every keyword, there is a different keyword with the same number of matching documents. The leakage function is now slightly modified to output the number of results, including fake documents, for a search query, giving the function  $\mathcal{L}_{len}^{\alpha-pad}$ , where  $\alpha$  will be the minimum size of clusters induced by  $\mathcal{L}_{len}^{\alpha-pad}$ . Then, from Proposition 7.3, we have that  $\mathfrak{C}^{\mathcal{DB}}$  is  $(\mathcal{L}_{len}^{\alpha-pad}, \alpha)$ -acceptable.

Yet, using the fact that  $\mathcal{L}_{len}^{\alpha-pad}(q)$  is only a function of DB and  $q$  (and does not depend on previous queries), we can show a more general result, also better in terms of security.

**Proposition 7.4.** *Let  $C$  be a constraint with  $k$  free queries (cf. Definition 7.2), and  $\mathcal{L}$  a leakage function such that  $\mathcal{L}(q)$  is a stateless function of DB and  $q$ . Then the clustering  $\Gamma_{\mathcal{L}}(H)$  only depends on DB. Let  $\alpha = \min_i |G_i|$ . Then  $C$  is  $(\mathcal{L}, \alpha^k)$ -acceptable.*

The idea behind Proposition 7.4 is that we can change any of the  $k$  free queries of an history  $H$  by picking a different query in the same cluster, without modifying the leakage nor making the  $C$  not satisfied. As the  $k$  queries are free and the leakage of each query does not depend on the other ones, we can combine all the possibilities and create  $\alpha^k$  histories with the same leakage.

Proposition 7.4 gives a security level that is a lot better than the one implied by Proposition 7.3:  $(\mathcal{L}, \alpha)$ -acceptability for Proposition 7.3 vs.  $(\mathcal{L}, \alpha^k)$ -acceptability for Proposition 7.4. This last proposition actually offers  $\log \alpha$  bits of security for each search query, while the first one only offered security for the whole history and not individual query.

Hence, we only have to design a padding algorithm to ensure the security of ORAM-based schemes against attackers with prior knowledge of the database. We present such an algorithm in Section 7.4.

**Result length and search pattern.** Unfortunately,  $\mathcal{L}_{len}$  leakage only covers not really practical solutions. Many of the existing static schemes also leak the *search pattern*, the repetition of search queries. It is similar to the  $L1$  leakage of [CGPR15], but  $L1$  is not result hiding and leaks searched keywords co-occurrence. We denote this leakage function  $L1_{RH}$ . In particular, we are no longer in the setting of Proposition 7.4 where the leakage is independent of the past queries. Also, even if we use padding to hide the results length (and end up with leakage function  $L1_{RH}^{\alpha-pad}$ ), Proposition 7.3 does not apply either: it is easy to construct an history such that the clustering  $\Gamma_{L1_{RH}^{\alpha-pad}, C^{\mathcal{DB}}}(H)$  has clusters containing only one query (for repeating queries).

However, we can still show  $\mathfrak{C}^{\mathcal{DB}}$  is an  $(L1_{RH}^{\alpha-pad}, \alpha)$ -acceptable set of constraints, where  $\alpha$  is the minimum cluster size (over all constructible databases). Indeed, as constraints in  $\mathfrak{C}^{\mathcal{DB}}$  leave all queries free, for every history  $H = (\text{DB}, q_1, \dots, q_n)$ , we can generate a different history  $H'$  with the same leakage by choosing another first query  $q \neq q_1$  matching the same number of documents, and changing all queries  $q_i = q_1$  to  $q$ . Also, if there are queries  $q_j = q$  in  $H$ , we switch them to  $q_1$ . This gives us a history  $H' \neq H$  with the same leakage as  $H$ , and as we have at least  $\alpha - 1$  choices for  $q$ , we infer the  $(L1_{RH}^{\alpha-pad}, \alpha)$ -acceptability.

Finally, it is interesting to notice that, if we force queries to be different (e.g. using a dedicated constraint), we can show a better result. Namely, let  $\mathfrak{C}^{\mathcal{DB}, k}$  be the set of constraints fixing the database and satisfied by histories with  $k$  search queries that are pairwise different. Then  $\mathfrak{C}^{\mathcal{DB}, k}$  is  $(L1_{RH}^{\alpha-pad}, \beta(\alpha, k))$ -acceptable where  $\beta(\alpha, k) = \frac{\alpha!}{(\alpha-k-1)!}$  for  $k \leq \alpha$  and  $\beta(\alpha, k) = \alpha!$  otherwise. This result might be surprising: we add more constraints, but we have a higher apparent security level. However this is consistent with our security definitions, yet counter-intuitive: indeed, adding the pairwise distinct queries constraint reduces the space of histories satisfying the constraints,

and somewhat artificially removed the histories that had only  $\alpha - 1$  matching histories with the same constraint. We can also interpret this as the fact that, previously, the adversary could have learned that the queries were not repeating, but as he already knows this information when using constraints in  $\mathcal{C}^{\mathcal{DB},k}$ , there is no problem in leaking it.

## 7.4 Application to Database Padding with Best Possible Security

In this section, we will show how to pad the database in order to achieve  $\mathcal{L}_{len}^{\alpha-pad}$ , as presented in section 7.3.2. In particular, we want to create clusters of minimal size  $\alpha$ , based on the number of matches of every search query.

We want to solve this problem not only for static databases, but also for dynamic ones. Also, we present here a *black-box* construction: we apply the countermeasure to scheme with  $\mathcal{L}_{len}$  (resp.  $L1_{RH}$ ) leakage (leaking only the result length - resp. the result length and the search pattern) without needing access to the inner machinery of the scheme, to turn them into  $\mathcal{L}_{len}^{\alpha-pad}$  (resp.  $L1_{RH}^{\alpha-pad}$ ) secure schemes. We only need the client to store a table with  $K$  entries, counting the occurrence of every keyword (note that many dynamic SE schemes already need  $\mathcal{O}(K)$  - or similar - permanent or transient storage [CJJ+14; SPS14; GMP16], *Σοφοϋς*, Diana, Janus, ...).

### 7.4.1 Using Frequencies Instead of Counts

To make our analysis easier and more general, we will work with keywords' frequency instead of exact result count. Namely, if the adversary knows the database (e.g. in a static setting), he will easily derive the frequencies, while our approach also covers an adversary with distributional knowledge of the database (typical in a dynamic setting without file injection attack). Also, we adopt a distributional approach, but, again in the case the adversary entirely knows the database, we can replace the expectancies by the actual real values.

So, let  $\mathcal{DB}$  be a distribution, with keywords in the set  $W$ . For  $DB \leftarrow \mathcal{DB}$  and  $w \in W$ , we recall that  $N_w = |DB(w)|$ , and  $N = |DB|$ . We note the expected frequency of  $w$  as  $e_w$ :

$$e_w := \mathbb{E}_{DB \leftarrow \mathcal{DB}} \left[ \frac{N_w}{N} \right].$$

$N_w$  follows a multinomial law of  $N$  trials with event probability  $e_w$ :

$$\sum_{w \in W} N_w = N \text{ and } \text{Var}(N_w) = N e_w (1 - e_w)$$

By applying Chebyshev's inequality to  $N_w$ , we have

$$\mathbb{P}[|f_w - e_w| \geq \varepsilon] \leq \frac{e_w(1 - e_w)}{N\varepsilon^2}. \quad (7.1)$$

where  $f_w := \frac{N_w}{N}$  is the real frequency of  $w$  in DB. In particular, Equation (7.1) tells us that the observed frequency converges towards the expected frequency as the database grows, and an adversary will be able to tell if a keyword  $w$  is a good candidate for a query matching  $n$  documents from the distance between the observed frequency  $f_w$  and the expected frequency  $e_w$  for  $w$ .

As mentioned in Section 7.3.2, to thwart count-based/frequency-based attacks, we want to pad the database so that, for every keyword  $w$ , there are at least  $\alpha - 1$  different keywords with the same

frequency in the database. Here, we are talking about the *observed* frequencies, *i.e.* the frequencies after padding, the ones the scheme will leak to the adversary. Also, once we know that the expected observed frequencies of different keywords are identical, Equation (7.1) ensures that the actual frequencies of these keywords will converge to the same value, and will be indistinguishable, forming a cluster with these keywords.

#### 7.4.2 How to Pad

In the following, we will denote the expected real (resp. observed) frequency of keyword  $w$  by  $e_w^r$  (resp.  $e_w^o$ ). The clusters will be formed by keywords with the same expected observed frequency:  $\Gamma(e^o) = (G_1, \dots, G_m)$  such that  $\exists(\tilde{e}_1, \dots, \tilde{e}_m)$  with  $G_i = \{w | e_w^o = \tilde{e}_i\}$ .

To achieve this, the client will pad the real keyword distribution by inserting fake entries (keyword/document pairs) whose keyword is chosen according to a *padding distribution*, a multinomial distribution of parameter  $e_w^p$ . More formally, when answering a query on keyword  $w$ , the server will see that it matches  $N_w^o$  documents which can be decomposed as

$$N_w^o = N_w^r + N_w^p \quad (7.2)$$

where  $N_w^r$  is the number of real documents matching  $w$  and  $N_w^p$  is the number of fake entries used for padding. Similarly, the total number of entries in the padded database,  $N^o$  can be decomposed as  $N^r + N^p$ , with  $N^r$  being the number of real entries and  $N^p$  the number of fake entries. As previously, we can express  $e_w^o$ ,  $e_w^r$ , and  $e_w^p$  – the parameter of the padding distribution – as

$$e_w^o = \mathbb{E} \left[ \frac{N_w^o}{N^o} \right], e_w^r = \mathbb{E} \left[ \frac{N_w^r}{N^r} \right], \text{ and } e_w^p = \mathbb{E} \left[ \frac{N_w^p}{N^p} \right].$$

We denote by  $\gamma$  the ratio of fake entries inserted in the database:

$$N^p = \gamma N^r$$

By combining this with Equation (7.2), we end up with the following relationship among the expected keyword frequency:

$$e_w^o = \frac{1}{1+\gamma} e_w^r + \frac{\gamma}{1+\gamma} e_w^p \Leftrightarrow e_w^p = \left(1 + \frac{1}{\gamma}\right) e_w^o - \frac{1}{\gamma} e_w^r. \quad (7.3)$$

For each new real entry added to the database, the client, knowing the distribution of the database, and with expected observed frequencies of his choice, will hence create, on average,  $\gamma$  padding entries with keywords chosen according to a categorical law of parameter  $(e_w^p)_{w \in W}$ .

Also, it is extremely important to notice that, as  $e_w^p$  must be comprised between 0 and 1, Equation (7.3) gives us a lower bound on  $\gamma$ :

$$\gamma \geq \max_{w \in W} \left\{ \frac{e_w^r - e_w^o}{e_w^o}, \frac{e_w^r - e_w^o}{e_w^o - 1} \right\}. \quad (7.4)$$

Hence, if the client does not want to add too many fake entries (to reduce the cost of padding), he cannot choose any expected observed frequency distribution.



**Practical considerations on how to insert fake entries in the database.** An important point to notice about  $\gamma$  and the way we do padding is that the adversary must not be able to distinguish fake entries from real ones when they are inserted. Otherwise, at least for structured encryption-based searchable encryption, the adversary will be able to filter between real and fake entries.

Also, if we suppose that the number of additional fake entries inserted per real entry is not constant, the adversary could mark the updates with a low number of fake entries and restrict the count-based attack to only these. This will help her to defeat the padding counter-measure more easily as the observed keyword frequency for this reduced subset of entries will be closer to the real frequency than for the entire database.

However, this does not prevent  $\gamma$  from being non-integral: for example to get  $\gamma = 1/3$ , instead of inserting a fake entry one time out of two, we could cache updates, wait for three of them to be available, and then send 4 entries (the three real ones and a fake one) to the server. Hence, in practice, we will choose a rational value for  $\gamma$ .

### 7.4.3 Constructing Frequency-based Clusters

In the previous section, we showed how the client could pad the database once he chose a target observed frequencies distribution that will form clusters. In this section, we will see how to construct this distribution in a way that minimizes  $\gamma$ , given the minimum cluster size  $\alpha$ .

Formally, by the end of this section, we would have described an algorithm that, on input a real keyword distribution  $(e_w^r)$  and a parameter  $\alpha$ , outputs an expected observed keyword frequency distribution  $(e_w^o)$  such that the clustering  $\Gamma(e^o)$  has clusters of size at least  $\alpha$ , and that the padding cost  $\gamma$  is minimized. This algorithm will run in time  $\Theta((K - \alpha)\alpha)$ .

**Cost metrics.**  $\gamma$  can be seen as the cost of the countermeasure. It measures the additional server storage induced by the padding. However, one could consider instead other cost metrics, such as the bandwidth overhead, or the computational overhead for search queries. Here, we only focus on the storage cost  $\gamma$ .

**Expected frequencies with minimal cost for a given clustering.** Constructing the clustering from the frequencies and optimizing the choice of these is not easy in practice because the process of computing  $\Gamma(e^o)$  from  $e^o$  is highly discontinuous and it is hard to predict what will happen to the sizes of clusters of  $\Gamma(e^o)$  when  $e$  is changed.

On the other hand, it is easier to construct the expected frequencies from a reasonable clustering choice. Namely, a frequency-based clustering  $\Gamma = (G_1, \dots, G_m)$  and the associated expected frequencies  $(\tilde{e}_i)$  (cf. Section 7.4.2) have to satisfy the equations

$$\sum_{i=1}^m |G_i| = K \quad (7.5)$$

$$\sum_{i=1}^m |G_i| \tilde{e}_i = 1. \quad (7.6)$$

Equation (7.6) comes from the fact that the frequencies  $e_w^o$  must sum to 1, and is obtained by regrouping keywords by cluster. Also, as we want to minimize the padding cost, we want  $\gamma$  to be as small as possible, and the minimum value for  $\gamma$  is, from Equation (7.4)

$$\gamma_{\min}(\Gamma, \tilde{e}) = \max_{w \in W} \left\{ \frac{e_w^r - e_w^o}{e_w^o}, \frac{e_w^r - e_w^o}{e_w^o - 1} \right\} = \max_{1 \leq i \leq m} \left\{ \max_{w \in G_i} \left\{ \frac{e_w^r - \tilde{e}_i}{\tilde{e}_i}, \frac{e_w^r - \tilde{e}_i}{\tilde{e}_i - 1} \right\} \right\} \quad (7.7)$$

For a given frequency-based clustering  $\Gamma$ , we can find the expected observed frequencies ( $\tilde{e}_i$ ) of keywords in each cluster minimizing the padding cost, as presented in Theorem 7.5.

**Theorem 7.5.** *For a cluster  $G_i$  of  $\Gamma$ , we denote  $e_{\max(i)}^r$  the maximum value of  $e_w^r$  for  $w \in G_i$ . The minimum value  $\gamma_{\min}(\Gamma)$  of  $\gamma_{\min}(\Gamma, \tilde{e})$  (over all possible cluster frequencies choices) is*

$$\gamma_{\min}(\Gamma) = \min_{(\tilde{e}_i)_{1 \leq i \leq m}} \gamma_{\min}(\Gamma, \tilde{e}) = \left( \sum_{i=1}^m |G_i| e_{\max(i)}^r \right) - 1 \quad (7.8)$$

and is attained for

$$(\tilde{e}_i)_{1 \leq i \leq m} = (\tilde{e}_i^*)_{1 \leq i \leq m} = \left( \frac{e_{\max(i)}^r}{1 + \gamma} \right),$$

i.e.  $\gamma_{\min}(\Gamma, \tilde{e}_i^*) = \gamma_{\min}(\Gamma)$ .

*Proof.* Using the above notations,  $\gamma_{\min}(\Gamma, \tilde{e})$  can be re-written as

$$\gamma_{\min}(\Gamma, \tilde{e}) = \max_{1 \leq i \leq m} \left\{ \frac{e_{\max(i)}^r}{\tilde{e}_i}, \frac{1 - e_{\min(i)}^r}{1 - \tilde{e}_i} \right\} - 1. \quad (7.9)$$

From this equation, we can easily derive a lower bound for  $\gamma$ : for each cluster,  $\gamma_i(\Gamma, \tilde{e}_i) = \max \left\{ \frac{e_{\max(i)}^r}{\tilde{e}_i}, \frac{1 - e_{\min(i)}^r}{1 - \tilde{e}_i} \right\} - 1$  is minimum when

$$\frac{e_{\max(i)}^r}{\tilde{e}_i} = \frac{1 - e_{\min(i)}^r}{1 - \tilde{e}_i} \Leftrightarrow \tilde{e}_i = \tilde{e}_i^* \text{ where } \tilde{e}_i^* := \frac{e_{\max(i)}^r}{1 + e_{\max(i)}^r - e_{\min(i)}^r}.$$

and, as a consequence,

$$\gamma_{\min}(\Gamma, \tilde{e}) \geq \gamma_0 = \max_{1 \leq i \leq m} \left\{ e_{\max(i)}^r - e_{\min(i)}^r \right\}. \quad (7.10)$$

A very important thing to notice is that, without loss of generality, we can suppose that either  $\tilde{e}_i \geq \tilde{e}_i^*$  for all  $i$ , or  $\tilde{e}_i \leq \tilde{e}_i^*$  for all  $i$ , when the optimal cost is reached. Suppose this is not the case, and that for the majority of the clusters  $\tilde{e}_i \geq \tilde{e}_i^*$ , and take  $j$  such that  $\tilde{e}_j < \tilde{e}_j^*$ . Then, by decreasing  $\tilde{e}_i > \tilde{e}_i^*$  and increasing  $\tilde{e}_j < \tilde{e}_j^*$  such that  $|G_i| \tilde{e}_i + |G_j| \tilde{e}_j$  remains constant, we will decrease both  $\gamma_i$  and  $\gamma_j$ . Then if the maximum of  $\gamma_i$ 's is reached for cluster  $G_i$ , or  $G_j$  (or both), this contradicts the cost minimality (the same argument holds when the optimal cost is reached for several clusters at the same time).

Before going on, we have to prove an helpful technical lemma, and give the following definitions:

$$\delta_w^+ = e_{\max(i)}^r - e_w^r \\ \Delta = \max_{w \in W} \delta_w^+.$$

**Lemma 7.6.** *The following inequalities hold:*

$$1 + \Delta \leq \sum_{i=1}^m |G_i| e_{\max(i)}^r \leq 1 + (K - m) \Delta \quad (7.11)$$

*Proof.* By definition of  $\delta_w^+$ ,

$$\begin{aligned} \sum_{i=1}^m \sum_{w \in G_i} e_w^r &= \sum_{i=1}^m \sum_{w \in G_i} (e_{\max(i)}^r - \delta_w^+) \\ \Leftrightarrow \sum_{i=1}^m |G_i| e_{\max(i)}^r &= 1 + \sum_{i=1}^m \sum_{w \in G_i} \delta_w^+ \\ \Rightarrow 1 + \Delta &\leq \sum_{i=1}^m |G_i| e_{\max(i)}^r \leq 1 + (K - m)\Delta. \end{aligned}$$

The  $K - m$  factor instead of  $K$  in the last inequality comes from the fact that in each cluster, there is at least one keyword  $w$  such that  $\delta_w^+ = 0$ .  $\square$

From this lemma, we can show that  $\sum_{i=1}^m |G_i| \tilde{e}_i^* \geq 1$ :

$$\sum_{i=1}^m |G_i| \tilde{e}_i^* = \sum_{i=1}^m |G_i| \frac{e_{\max(i)}^r}{1 + \Delta_i} \geq \sum_{i=1}^m |G_i| \frac{e_{\max(i)}^r}{1 + \Delta} \geq 1$$

However, we are not guaranteed that  $\sum_{i=1}^m |G_i| \tilde{e}_i^* = 1$ , and we cannot conclude that the optimal cost will be  $\gamma_{\min}$ .

Suppose  $\sum_{i=1}^m |G_i| \tilde{e}_i^* > 1$ . We need to decrease some values  $\tilde{e}_i$  to satisfy constraint (7.5). It is 'free' – it does not increase the overall cost – to do so for clusters such that  $\gamma_i < \gamma_{\min}$ , and for such clusters the minimum value that  $\tilde{e}_i$  can take is  $\tilde{e}_i^{\text{lim}}$  such that

$$\frac{e_{\max(i)}^r}{\tilde{e}_i^{\text{lim}}} = 1 + \gamma_{\min} \Leftrightarrow \tilde{e}_i^{\text{lim}} = \frac{e_{\max(i)}^r}{1 + \gamma_{\min}} = \frac{e_{\max(i)}^r}{1 + \Delta}.$$

Indeed as,  $\tilde{e}_i^{\text{lim}} \leq \tilde{e}_i^*$ ,  $\frac{e_{\max(i)}^r}{\tilde{e}_i^{\text{lim}}}$  is larger than  $\frac{1 - e_{\min(i)}^r}{1 - \tilde{e}_i^{\text{lim}}}$ .

Again, using Equation (7.11), we can show that  $\sum_{i=1}^m |G_i| \tilde{e}_i^{\text{lim}} \geq 1$ . Thus we will have to chose  $\tilde{e}_i < \tilde{e}_i^{\text{lim}}$  for some clusters. Also, for the optimal expected frequencies (the ones inducing the smallest cost), we will have  $\gamma_i = \gamma$  for all clusters: if this is not the case, and that there is a cluster such that  $\gamma_j < \gamma$ , we can decrease  $\tilde{e}_j$  (and thus increase  $\gamma_j$ ) while increasing the other expected frequencies (and thus decreasing  $\gamma$ ). Hence, from the definition of  $\gamma_i$ , we have that

$$\gamma = \frac{e_{\max(i)}^r}{\tilde{e}_i} - 1 \Leftrightarrow \tilde{e}_i = \frac{e_{\max(i)}^r}{1 + \gamma}$$

and as  $\tilde{e}$  satisfies the constraint (7.6), we get

$$\sum_{i=1}^m |G_i| \frac{e_{\max(i)}^r}{1 + \gamma} = 1 \Leftrightarrow \gamma = \left( \sum_{i=1}^m |G_i| e_{\max(i)}^r \right) - 1. \quad (7.12)$$

$\square$

Theorem 7.5 directly gives us an algorithm, that, given a clustering  $\Gamma$ , finds the keyword frequencies minimizing the padding cost, together with this cost  $\gamma_{\min}(\Gamma)$ . As a consequence, to find the expected observed frequencies distribution that minimizes the padding cost, and that induces clusters of size at least  $\alpha$ , we only have to find the clustering of keywords inducing the minimum cost and deriving the frequencies  $(\tilde{e}_i)$ .

**Finding a clustering with minimal cost.** The last step is to design an algorithm that, given expected real keyword frequencies ( $e_w^r$ ), and parameter  $\alpha$ , constructs a clustering  $\Gamma$  with a minimum cost  $\gamma_{\min}(\Gamma)$  with the constraint that every cluster has size at least  $\alpha$ .

To simplify the problem, we can notice that, if  $\Gamma = (G_1, \dots, G_m)$  reaches the minimum cost, and that there are two clusters  $G_i$  and  $G_j$  such that  $e_{\max(i)}^r = e_{\max(j)}^r$ , then we can ‘merge’ these two clusters and construct a clustering with the same cost (from Equation (7.8)), whose clusters size is larger than  $\alpha$ . Hence, we can suppose that  $e_{\max(i)}^r \neq e_{\max(j)}^r$  for every pair of clusters.

Also, for such a clustering reaching the minimum cost, we can suppose that

$$\forall G_i, G_j \in \Gamma, e_{\max(i)}^r \leq e_{\min(j)}^r \text{ or } e_{\min(i)}^r \geq e_{\max(j)}^r. \quad (7.13)$$

An equivalent (but more verbose) definition would be

$$\forall G_i, G_j \in \Gamma, (\forall w \in G_i, w' \in G_j, e_w^r \leq e_{w'}^r) \text{ or } (\forall w \in G_i, w' \in G_j, e_w^r \geq e_{w'}^r),$$

justifying that we call such clusterings *monotone*. We can easily show the following proposition.

**Proposition 7.7.** *Let  $\Gamma$  be a non-monotone clustering with clusters of size at least  $\alpha$ . Then there exists a clustering  $\Gamma'$  with clusters of size at least  $\alpha$ ,  $\gamma_{\min}(\Gamma) \geq \gamma_{\min}(\Gamma')$  and  $\Gamma'$  is monotone.*

*Proof.* As  $\Gamma$  is not monotone, there are two clusters  $C_i$  and  $C_j$  such that  $e_{\max(i)}^r > e_{\max(j)}^r > e_{\min(i)}^r$ . Let  $w \in C_i$  (resp.  $w' \in C_j$ ) such that  $e_w^r = e_{\min(i)}^r$  (resp.  $e_{w'}^r = e_{\max(j)}^r$ ), and  $\Gamma'$  be the clustering obtained by exchanging  $w$  and  $w'$  in  $C_i$  and  $C_j$ . If there is only one  $w' \in C_j$  reaching the maximum, The maximum expected frequency of the new cluster  $C_j'$  will decrease or stay identical if there is more than one  $w' \in C_j$  reaching the maximum, while the  $e_{\max(i)}^r$  will not change. As a consequence,  $\gamma(\Gamma) \geq \gamma(\Gamma')$ .

We can iterate this algorithm until  $C_i$  and  $C_j$  satisfy the monotonicity condition, *i.e.* with  $e_{\max(j)}^r \leq e_{\min(i)}^r$ . Finally, by repeating this procedure over pairs of clusters, as in a sorting algorithm, we end up constructing a monotone clustering whose cost is less than the original one, while its min-quality remains unchanged (because the size of the clusters did not change).  $\square$

We now have to simple constraints on the clustering we want to construct: first its clusters are of size at least  $\alpha$ , then the clustering should be monotone. Also, it will be handy in the following to suppose that the keywords are numbered such that their expected frequencies are non increasing: for  $i < j$ ,  $e_{w_i}^r \leq e_{w_j}^r$ . Constructing such a clustering is equivalent to fixing  $m' = m - 1$  cluster limits  $\ell_0 = 0 < \ell_1 < \dots < \ell_m < K = \ell_{m+1}$  such that

$$\forall 1 \leq i \leq m + 1, \ell_i - \ell_{i-1} \leq \alpha.$$

and the associated clustering is  $\Gamma_\ell = (G_1, \dots, G_m)$  with  $G_i = \{w_j | \ell_{i-1} < j \leq \ell_i\}$ .

We could try to enumerate all these clusterings to find the clustering with minimum cost. However, the number of such clusterings is lower bounded by the number of partitions  $p(K)$  of  $K$  (there is a surjective mapping between these clusterings and the values taken by  $\ell_i - \ell_{i-1}$ ), and  $p(K)$  grows extremely fast as, from Hardy and Ramanujan [HR18],

$$p(n) \sim \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right).$$

Instead, we can reduce the problem of finding the optimal clustering to the problem of finding a shortest path in an directed acyclic graph (DAG) of  $K + 1$  vertices and  $\frac{(K-\alpha)(K-\alpha+1)}{2}$  edges. Finding

a solution to this problem can be done very efficiently, in in time complexity  $\Theta((K - \alpha)^2)$  and memory complexity  $\Theta(K)$  [CLRS09].

To do the reduction, we consider the graph  $G_\alpha(W) = (V, E)$  with vertices  $V$  and edges  $E$  defined as follows:

$$\begin{aligned} V &= \{0, \dots, K\} \\ E &= \{(i, j) \in V^2 \mid j - i \geq \alpha\}. \end{aligned}$$

The weight  $c(i, j)$  of the edge  $(i, j) \in E$  is defined as

$$c(i, j) := (j - i)e_{w_i}^r.$$

We can see that there is a mapping between path of  $G_\alpha(W)$  from node 0 to node  $K$  and monotone clusterings of minimal cluster size larger  $\alpha$ : for such a clustering, with cluster limits  $0 = \ell_0 < \ell_1 < \dots < \ell_{m-1} < \ell_m = K$ , we consider the path  $P_\Gamma = (\ell_0, \dots, \ell_m)$ . This path goes from 0 to  $K$  and two consecutive nodes in this path have a difference of at least  $\alpha$ , as its edges are all in  $E$ . Also, the weight of  $P_\Gamma$  is exactly  $\gamma_{\min}(\Gamma) + 1$  following from the definition of the edges' weight and from Equation (7.8):

$$\gamma_{\min}(\Gamma) + 1 = \sum_{i=1}^m |G_i| e_{\max(i)}^r = \sum_{i=1}^m |\ell_i - \ell_{i-1}| e_{w_{\ell_i}}^r = c(P_\Gamma).$$

So, minimizing the clustering cost is equivalent to finding a shortest path in  $G_\alpha(W)$ , which is clearly a DAG with  $K + 1$  vertices and  $(K - \alpha)(K - \alpha + 1)/2$  edges.

We can reduce also the number of edges to consider in the graph to  $|E| \leq (K - \alpha)\alpha$ , reducing the computational complexity of the clustering algorithm to  $\Theta((K - \alpha)\alpha)$  – the algorithm is now linear in  $K$ .

**Reducing the complexity of the clustering graph.** We can reduce the number of edges to consider in the graph to  $|E| \leq (K - \alpha)\alpha$ , reducing the computational complexity of the clustering algorithm to  $\Theta((K - \alpha)\alpha)$  – the algorithm is now linear in  $K$ .

Suppose that  $\Gamma$  be a clustering with minimum cost, whose clusters are larger than  $\alpha$ , and such that no cluster contains more than  $2\alpha$  keywords. We can construct a clustering  $\Gamma'$  by splitting a cluster of size larger than  $2\alpha$  in two clusters each of size larger than  $\alpha$ .  $\Gamma'$  will have a cost less than  $\Gamma$  (because the maximum expected frequency of the newly obtained cluster will be less than the one of the old split cluster). Without loss of generality, we can suppose that minimal cost clusterings have no cluster with more than  $2\alpha$  elements.

This implies that we can reduce the set of edges of  $G$  to

$$E = \{(i, j) \in V^2 \mid j - i \geq \alpha \text{ and } j - i < \alpha\}.$$

The number of edges is then  $|E| = (K - 2\alpha)\alpha + \frac{\alpha(\alpha-1)}{2}$  ( $\alpha$  incoming edges for all nodes, except the first  $\alpha$  ones – no incoming edge – and the nodes  $i \in [\alpha, 2\alpha - 1]$  which have  $i - \alpha$  incoming edges).

Finally, note that as described higher, the graph  $G$  is already topologically sorted. Hence, finding the shortest path in  $G$  can be very easily done using dynamic programming.

#### 7.4.4 Integration to Existing Schemes

The padding algorithm described in the previous sections ensures that, for a given input parameter  $\alpha$ , there are clusters, each of size at least  $\alpha$ , of keywords with the same (adversarially observed) frequency. This algorithm can be integrated to any SE scheme very simply: the client, during the setup phase, as he knows the upcoming database distribution, is able to compute the clustering and the padding keyword distribution. As explained in Section 7.4.2, we can suppose that  $\gamma$  is a rational number  $\gamma = p/q$ . Hence, the client will keep a buffer of  $q$  entries to be pushed to the server.

When a new entry has to be inserted, the client inserts it in the buffer. If the buffer is full, he creates  $p$  fake entries by sampling  $p$  random keywords according to the padding distribution and chooses special document indices marking that these entries are fake. Finally, he pushes the  $q$  real and  $p$  fake entries to the server, without forgetting to randomly permute them beforehand.

This construction can directly be applied to searchable encryption schemes like  $\Pi_{\text{bas}}^{\text{dyn}}$  [CJJ+14], SPS [SPS14], TWORAM [GMP16],  $\Sigma\phi\phi\phi$  (Section 4.5), Diana (Section 4.6), and many others to transform them into schemes that are provably secure against adversaries with knowledge of the database, as a corollary of Section 7.3.2.

**Corollary 7.8.** *Let  $\mathcal{C}_k^{\text{DB}}$  the set of database-fixing constraints (cf. Section 7.2.3) with  $k$  search queries. Used with the padding algorithm with parameter  $\alpha$ , TWORAM is  $(\mathcal{L}_{\text{len}}^{\alpha\text{-pad}}, \mathcal{C}_k^{\text{DB}}, \alpha^k)$ -constrained-adaptive-indistinguishability secure.*

**Corollary 7.9.** *Used with the padding algorithm with parameter  $\alpha$ ,  $\Pi_{\text{bas}}^{\text{dyn}}$ , SPS,  $\Sigma\phi\phi\phi$ , Diana in a result-hiding scenario, are  $(L_{RH}^{\alpha\text{-pad}}, \mathcal{C}^{\text{DB}}, \alpha)$ -constrained-adaptive-indistinguishability secure.*

## 7.5 Experiments

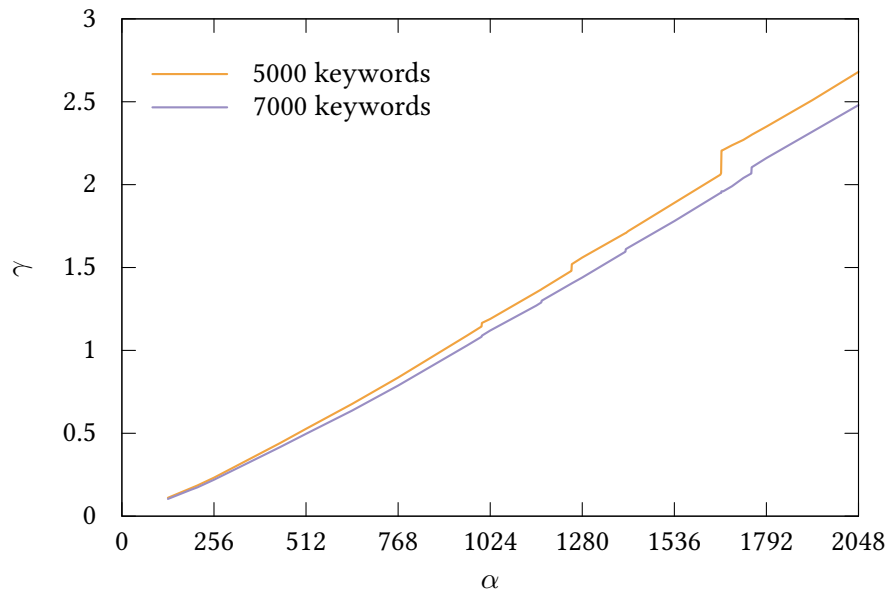
We implemented the frequency-based clustering algorithm. This allowed us to study the influence of the parameter  $\alpha$  on the cost of the clustering, the computational overhead of the padding (the computation of the clustering and of the padding distribution). Finally, our experiments show that the count attack of Cash *et al.* is a lot more powerful than originally assessed in their paper [CGPR15]. In particular, when the adversary knows the database, leaking the keywords co-occurrence is devastating.

### 7.5.1 The Performance of the Clustering Algorithm

We implemented the clustering algorithm in Java and ran it on the Enron dataset [Enron] for different values of  $\alpha$ , in order to see the relation between  $\alpha$  and  $\gamma$ . The obtained results are summarized in Figure 7.4. We can see that the cost grows roughly as the minimum cluster size, although we can see some discontinuities. These singularities appear when the number of clusters of the optimal clustering changes. Namely, in the experiment with 5000 keywords, the algorithm with  $\alpha = 1666$  generates 3 clusters, while for  $\alpha = 1667$ , it only generates 2.

As we would expect, for a given database, the cost of the clustering decreases as the number of indexed keywords increases: the clustering algorithm has more choices to construct clusters and hence will optimize this choice more easily.

Also, the clustering algorithm itself is quite fast: for 20 000 keywords and  $\alpha \approx 4750$ , the algorithm runs in about 170 ms using a quite naive dynamic programming implementation, on an Intel 4980HQ CPU running at 2.80 GHz. In comparison,  $\Sigma\phi\phi\phi$  takes around 1.69 ms per insertion, and if we suppose that every document contains more than 100 keywords, we can recompute a clustering every



**Figure 7.4** – Storage overhead  $\gamma$  due to the database padding depending on  $\alpha$ .

10 (or 100) inserted documents without reducing noticeably the update throughput and prevent unnoticed changes in the database distribution. For dynamic schemes with higher throughput (e.g.  $\Pi_{\text{bas}}^{\text{dyn}}$  [CJJ+14], or Diana), this can be done every 1000 new document insertion to avoid hindering the update throughput.

### 7.5.2 Influence of Secure Padding on the Count Attack

As an experiment, we also run the count attack of Cash *et al.* against schemes with complete  $L1$  leakage – in particular co-occurrence leakage, not only  $L1_{RH}$  – to which we applied our padding algorithm. Remember that our frequency-based clustering provably protects schemes with  $L1_{RH}$  leakage. This experiment can be seen as a way to understand what actually is the security loss due to the co-occurrence information.

We executed the count attack on the Enron email database once padded with our algorithm, with different values for  $\alpha$ . Experimental results are stated in Table 7.1. For the experiments, we measured the elapsed CPU time: as the count attack is massively parallelizable, the wall clock time is not a good measurement, and the CPU time is representative of the actual cost for the attacker. We ran the attack using several randomness seeds, and we give both the average and the minimum running time over the choice of seeds. We also compared the attack running time in presence of two different padding strategies: our frequency-based clustering technique and the technique described in [CGPR15], namely the number of entries matching a keyword is padded up to the nearest multiple of an integer  $n$ , the *padding factor*.

The first thing to notice is that we always were able to recover (almost) all the queries. The original [CGPR15] paper showed partial reconstruction rate due to a small bug in their original code (premature exit of a loop exhausting the possible candidates for a guessed keyword). The count attack is much more powerful than originally stated in the original paper.

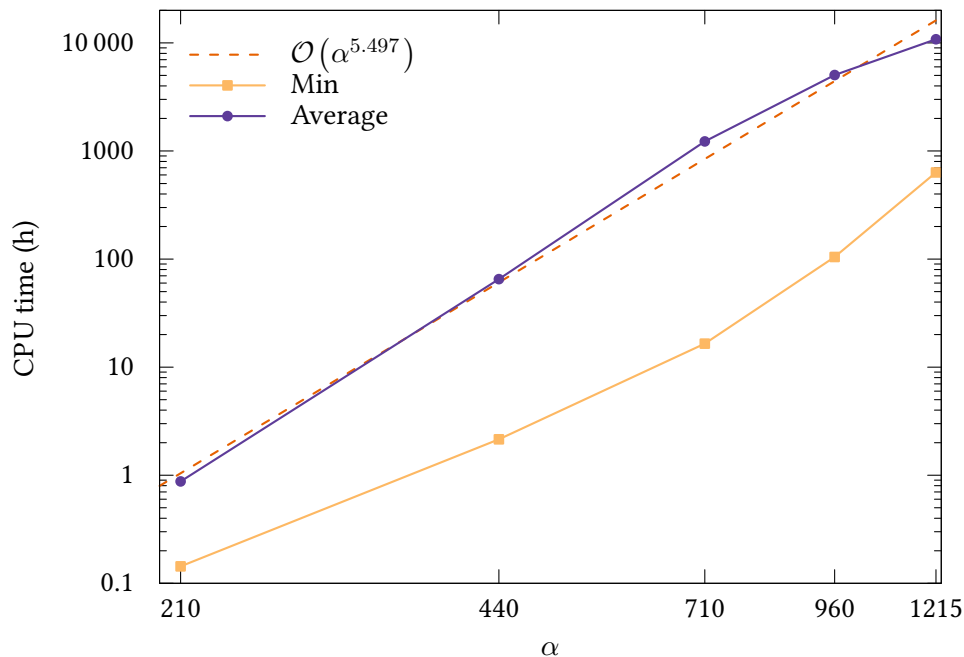
Yet, we can clearly see that the padding strategy influences quite a lot the performance of the count attack. We interpret that as a consequence of the way the count attack works. When there

**Table 7.1** – Experimental running time of the count attack with two types of padding. Min and average are taken over at least 20 runs. The Enron email dataset [Enron] was used for this experiment, with 5000 keywords and 500 queried keywords.  $n$  is the padding factor of the padding technique of [CGPR15].

$\alpha$	Clustering		Runtime blowup		Up to Multiple		
	Runtime (h)		Min	Avg.	Runtime (h)		$n$
	Min	Avg.			Min	Avg.	
210	0.143	0.876	$1.37 \times$	$3.61 \times$	0.105	0.243	100
440	2.15	65.2	$5.63 \times$	$22.5 \times$	0.382	2.90	200
710	16.5	1169	$13.8 \times$	$25.6 \times$	1.19	22.5	300
960	104	5040	$25.1 \times$	$17.3 \times$	4.17	292	400

are keywords with unique result count, the attacker is able to recover queries on these keywords and then uses this initial information to recover the other queries by comparing the number co-occurrence between a known and an unknown keyword on one side, and the known co-occurrence information he has from the knowledge of the dataset on the other side. For a still encrypted query  $q$ , the adversary will enumerate the candidate keywords matching the same number of documents and reject keywords with incompatible co-occurrence frequency with already recovered queries.

When this is not the case, the attacker cannot leverage the fact that he already recovered queries to compare the co-occurrence. So, instead he guesses the keyword corresponding to the target query among the  $\alpha$  and tries to proceed with the query recovery using this guess.



**Figure 7.5** – Running time of the count attack in presence of padding, depending on  $\alpha$ , in log-log scale.



So, in the latter case, we have that the count attack runtime should scale quadratically with  $\alpha$ . In practice, using a linear regression, we can see in Figure 7.5 that the attack scales more like  $\alpha^{5.5}$  in average, but we do not explain this discrepancy. In any case, we cannot rely on a sufficiently large value of  $\alpha$  to be out of reach of the count attack against the  $L1$  leakage. Just padding the database to have many keywords with the frequency, as the algorithm described in this paper does, *cannot* result in reasonable level of security, and there is a clear security gap between schemes leaking only the result length ( $\mathcal{L}_{len}$  leakage) and the one also leaking the keywords co-occurrence ( $L1$  leakage). We need a different approach to protect these schemes.

One is sketched by Islam *et al.* [IKK12, Sections 10-12] who introduced these co-occurrence attacks, and is, in spirit, similar to our clustering idea. Their idea is to pad the database so that there is a partition of the keyword set for which in each set, every keyword matches the same documents, and each set is at least of size  $\alpha$ . Unfortunately, their experiments show that the overhead due to this countermeasure is very high ( $\gamma \approx 2$  for  $\alpha = 2$ , and  $\gamma \approx 4$  for  $\alpha = 3$ ). Also, doing this in a dynamic setting would require up to  $\mathcal{O}(K^2)$  storage on the client side (in order to store the co-occurrence matrix), which would not scale for large databases.

Finally, we can see the attacks based on co-occurrence as an order 2 version of the frequency-based attack: instead of counting the number of total occurrence of a single-keyword  $w$  in the database, the adversary counts the number of occurrences of pairs  $(w_1, w_2)$ . Thus we can imagine higher order attacks using the co-occurrence information of 3 (or more) keywords in the same document. Provably thwarting this kind of attacks would be very costly in general, and we believe that we should actually rely on actual attack performance to evaluate the security of result-revealing SE schemes.

## References

- [BF17] Raphael Bost and Pierre-Alain Fouque. *Thwarting Leakage Abuse Attacks against Searchable Encryption – A Formal Approach and Applications to Database Padding*. Cryptology ePrint Archive, Report 2017/1060. <http://eprint.iacr.org/2017/1060>. 2017 (cit. on pp. x, 13, 151).
- [CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions*. In: *ACM CCS 06*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, Oct. 2006, pp. 79–88 (cit. on pp. 9, 42–45, 56, 72, 152).
- [CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. *Leakage-Abuse Attacks Against Searchable Encryption*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 668–679 (cit. on pp. 60, 66, 151–153, 156, 161, 162, 170–172).
- [CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. 10, 49, 56, 73, 83, 99, 129, 163, 170, 171).
- [CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 9780262033848 (cit. on p. 169).
- [Enron] Enron Corporation. *Enron email dataset*. [Link](#). (Cit. on pp. 170, 172).

- [GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. *TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption*. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563–592 (cit. on pp. 7, 8, 45, 69, 73, 99, 101, 161, 163, 170).
- [HR18] Godfrey H. Hardy and Srinivasa Ramanujan. “Asymptotic formulæ in combinatory analysis”. In: *Proceedings of the London Mathematical Society* 2.1 (1918), pp. 75–115 (cit. on p. 168).
- [IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. *Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation*. In: *NDSS 2012*. The Internet Society, Feb. 2012 (cit. on pp. 59, 60, 151, 152, 173).
- [KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O’Neill. *Generic Attacks on Secure Outsourced Databases*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1329–1340 (cit. on pp. 151–153, 160).
- [KO97] Eyal Kushilevitz and Rafail Ostrovsky. *Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval*. In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 364–373 (cit. on pp. 6, 153).
- [SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. viii–x, 10, 13, 45, 59, 67, 71, 73, 81, 97–99, 115, 119, 122, 130–133, 141, 147, 163, 170).
- [ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. *All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption*. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 2016, pp. 707–720 (cit. on pp. 10, 12, 60, 66, 76, 114, 152, 153, 156, 157).



A conclusion is the place where you get  
tired of thinking.

ARTHUR BLOCH

# Conclusion 8



THIS THESIS PRESENTED new results and new constructions of searchable encryption. This very last chapter summarizes the results presented previously, and tries to open new research directions to look at.

## 8.1 Summary of the Results

In this thesis, we presented the concepts and goals of searchable encryption, as well as constructions achieving several security/performance/features tradeoffs.

After having shown that, even for static schemes, security comes at an unavoidable cost in Theorem 3.2, we introduced two security features: forward privacy (Chapter 4) and backward privacy (Chapter 5), the former protecting the confidentiality of the updates, and the latter the secrecy of deleted entries.

In Theorem 4.1, we showed that forward privacy had a cost, either in terms of client storage or of update efficiency. We then studied three ways to construct forward-private schemes, either from static SSE schemes (Section 4.4), using a trapdoor permutation ( $\Sigma\phi\phi\phi$  – Section 4.5), or with a range-constrained pseudorandom function (Diana – Section 4.6). Each of these construction has its interest: generic construction from static SSE allows for improved locality or improved asymptotic complexity of the search protocol in presence of deletions, while  $\Sigma\phi\phi\phi$  is an optimally efficient scheme and Diana has the best practical efficiency. We presented the implementation of  $\Sigma\phi\phi\phi$  and Diana, and showed the practicality of these schemes.

In Chapter 5, we studied thoroughly backward privacy, by first giving formal security definitions, and then proposing 4 schemes, Fides, Moneta, Diana<sub>del</sub> and Janus, achieving several flavors of backward privacy, and several performance tradeoffs, Moneta being the most secure, but impractical, Fides being the simplest scheme, but with high communication overhead, Diana<sub>del</sub> being more efficient than Fides, but also needing a lot of communication, and Janus only needing a single round-trip and achieving optimal asymptotic communication complexity, but with a big computational overhead. In particular, we demonstrated that the practical efficiency of Janus would prevent it from performing well on real world databases.

We also studied verifiable SSE in Chapter 6, where, after showing a logarithmic lower bound on the efficiency of such constructions (Theorem 6.2), we described the first optimal dynamic SSE scheme (Section 6.3) using verifiable hash tables and incremental set hashing functions. We also showed how to easily turn  $\Sigma\phi\phi\phi$ , Diana and Janus into verifiable schemes, by making the client locally store an (incremental) hash of the result set. We ended that chapter by fixing the SPS scheme, and making it verifiable with a very low overhead compared to the non-verified version.

Finally, in Chapter 7, we tried to understand the leakage abuse attacks and presented new definitions capturing these attacks. We developed a framework, that we hope will be useful to devise new counter-measures, but also to find new attacks. In particular, we constructed a padding scheme

provably preventing attacks using the frequency of keywords to break the queries' secrecy.

## 8.2 Open Problems

Even though some interrogations about searchable encryption were answered by the previously presented results, many very interesting questions remain open, or were even raised by those results. We will try here to give a glimpse on some of these problems.

**Lower bound on the locality of forward private schemes.** In Section 4.3.2, we explained why it looks hard to construct a forward private scheme that has both good locality, and low search or update overhead. We saw how to actually construct forward private scheme with  $\mathcal{O}(\log N)$  locality,  $\mathcal{O}(\log N \log \log N)$  read efficiency,  $\mathcal{O}(\log N)$  search complexity and  $\mathcal{O}(\log^2 N)$  update complexity (cf. Section 4.4).

Yet, it would be very interesting to exhibit a lower bound on the locality of such forward private constructions, possibly derived from the lower bound on ORAM locality of [ACN+17].

**Better support of deletions.** In Chapter 5, we studied several constructions of dynamic SSE supporting deletions in a secure way. However, we did not solve the efficiency problems of SSE supporting deletions. Although it is easy to construct such schemes, having a good level of security for these is not easy: the most efficient forward private deletion-able scheme is SPS, whose search complexity can depend only on the number of actual matches of a keyword, instead of depending on the number of historically added documents matching this keyword. Also, it would be nice to construct a forward private scheme efficiently supporting the deletion of an entire document at once, without having to delete the corresponding keyword/document pairs one by one.

The case of backward-private schemes is even worse: Fides, Diana<sub>del</sub> and Janus present several tradeoffs between security and efficiency. For example, Janus search complexity,  $\mathcal{O}(n_w \cdot d_w)$ , is worse (both in theory and in practice) than Diana<sub>del</sub>'s  $\mathcal{O}(a_w)$ , but Janus only needs a single round-trip for the execution of the search protocol while Diana<sub>del</sub> requires two, and has a larger communication overhead (cf. Table 5.1).

This leads to the question of the existence of a lower bound on the search/update efficiency of a backward-private scheme. Indeed, we believe that the asymptotic efficiency of Janus is optimal for single round-trips schemes, but the formalization of the lower bound has to be quite complex as Diana<sub>del</sub> shows that, when more than one round-trip is allowed, one can do better than Janus.

**Counter measure against file injections.** In Section 7.4, we showed how to counter attacks based on keywords frequency. Unfortunately, this is far from being enough against active adversaries, able to insert forged documents in the database in order to break the confidentiality of the queries, *i.e.* able to mount file injection attacks. We saw in Chapter 4 how to thwart the adaptive versions of these attacks, which are often devastating. However, we did not solve the problem of non-adaptive file injection attacks.

These attacks, as mentioned in Section 7.2.3, can be studied under the framework of Chapter 7, so as to devise new counter measures. In particular, it seems that some kind of adaptive padding, that tries to hide when entries corresponding to a newly inserted document are actually pushed to the server, is absolutely necessary. Understanding how to implement this adaptive padding, and assessing its actual efficiency against an attacker looks to be a very interesting, yet very hard problem.

**Use of secure hardware.** It would also be interesting to study how to use secure hardware or secure enclaves, such as Intel SGX [Int17], to improve the performance of the SSE schemes. Indeed, the secure enclave could perform exactly the same operations than the client, but physically on the server, reducing the cost of interactions due to high latency between the (real) client and the server.

However, this would require a lot of confidence in the implementation of the protocols in the secure enclave (the server could use some side channels, and learn information that would have been hidden if the computations were run on the client), and even some confidence in the enclave itself. Instead, we could imagine a three-party model, as in [BO15], where the secure hardware is there to help the server to run some computation, using a client-issued key to decrypt useful information, but cannot learn everything about the results, using this key: the enclave and the server would have to collude to break the security to the scheme.

More generally, we could try to design schemes using such secure enclaves, and whose security is not completely lost when the enclave is itself broken. One example would be the use of secure attestations for verifiable SSE: the enclave would not have to store any secret to prove the integrity of the results of a search query.





# Bibliography

- [ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. *Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier*. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 843–862 (cit. on p. 7).
- [ABFK14] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. *XPIR: Private Information Retrieval for Everyone*. Cryptology ePrint Archive, Report 2014/1025. <http://eprint.iacr.org/2014/1025>. 2014 (cit. on p. 6).
- [ACN+17] Gilad Asharov, T-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. *Oblivious Computation with Data Locality*. Cryptology ePrint Archive, Report 2017/772. <http://eprint.iacr.org/2017/772>. 2017 (cit. on pp. 69, 178).
- [AG] D. F. Aranha and C. P. L. Gouvêa. *RELIC is an Efficient Library for Cryptography*. [Link](#). (Cit. on p. 115).
- [AKST14] Daniel Apon, Jonathan Katz, Elaine Shi, and Aishwarya Thiruvengadam. *Verifiable Oblivious Storage*. In: *PKC 2014*. Ed. by Hugo Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 131–148 (cit. on p. 130).
- [ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. *Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations*. In: *48th ACM STOC*. Ed. by Daniel Wichs and Yishay Mansour. ACM Press, June 2016, pp. 1101–1114 (cit. on pp. 58, 59, 69, 71).
- [BB04] Dan Boneh and Xavier Boyen. *Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles*. In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 223–238 (cit. on p. 28).
- [BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O’Neill. *Deterministic and Efficiently Searchable Encryption*. In: *CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. LNCS. Springer, Heidelberg, Aug. 2007, pp. 535–552 (cit. on p. 41).
- [BC15] Alexandra Boldyreva and Nathan Chenette. *Efficient Fuzzy Search on Encrypted Data*. In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Heidelberg, Mar. 2015, pp. 613–633 (cit. on p. 11).
- [BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *Keying Hash Functions for Message Authentication*. In: *CRYPTO’96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 1–15 (cit. on p. 32).

- [BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O’Neill. *Order-Preserving Symmetric Encryption*. In: *EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, Heidelberg, Apr. 2009, pp. 224–241 (cit. on p. 11).
- [BDJR97] Mihir Bellare, Anand Desai, Eric Jorjani, and Phillip Rogaway. *A Concrete Security Treatment of Symmetric Encryption*. In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 394–403 (cit. on p. 34).
- [BDOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. *Public Key Encryption with Keyword Search*. In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 506–522 (cit. on p. 41).
- [BEG+91] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. *Checking the Correctness of Memories*. In: *32nd FOCS*. IEEE Computer Society Press, Oct. 1991, pp. 90–99 (cit. on p. 120).
- [BF17] Raphael Bost and Pierre-Alain Fouque. *Thwarting Leakage Abuse Attacks against Searchable Encryption – A Formal Approach and Applications to Database Padding*. Cryptology ePrint Archive, Report 2017/1060. <http://eprint.iacr.org/2017/1060>. 2017 (cit. on pp. x, 13, 151).
- [BFP16] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. *Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security*. Cryptology ePrint Archive, Report 2016/062. <http://eprint.iacr.org/2016/062>. 2016 (cit. on pp. ix, 11, 13, 119).
- [BGH+13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. *Private Database Queries Using Somewhat Homomorphic Encryption*. In: *ACNS 13*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, Heidelberg, June 2013, pp. 102–118 (cit. on p. 6).
- [BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. *Functional Signatures and Pseudorandom Functions*. In: *PKC 2014*. Ed. by Hugo Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 501–519 (cit. on p. 30).
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *(Leveled) fully homomorphic encryption without bootstrapping*. In: *ITCS 2012*. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325 (cit. on p. 6).
- [BKOS07] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. *Public Key Encryption That Allows PIR Queries*. In: *CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. LNCS. Springer, Heidelberg, Aug. 2007, pp. 50–67 (cit. on p. 41).
- [BL16] Karthikeyan Bhargavan and Gaëtan Leurent. *Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH*. In: *NDSS 2016*. The Internet Society, Feb. 2016 (cit. on p. 10).
- [BM97] Mihir Bellare and Daniele Micciancio. *A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost*. In: *EUROCRYPT’97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, Heidelberg, May 1997, pp. 163–192 (cit. on p. 33).

- [BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. *Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives*. In: *ACM CCS 17*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1465–1482 (cit. on pp. [viii](#), [ix](#), [10](#), [13](#), [65](#), [97](#)).
- [BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. *The Round Complexity of Secure Protocols (Extended Abstract)*. In: *22nd ACM STOC*. ACM Press, May 1990, pp. 503–513 (cit. on p. [7](#)).
- [BN06] Paulo S. L. M. Barreto and Michael Naehrig. *Pairing-Friendly Elliptic Curves of Prime Order*. In: *SAC 2005*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. LNCS. Springer, Heidelberg, Aug. 2006, pp. 319–331 (cit. on p. [115](#)).
- [BO15] Foteini Baldimtsi and Olga Ohrimenko. *Sorting and Searching Behind the Curtain*. In: *FC 2015*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. LNCS. Springer, Heidelberg, Jan. 2015, pp. 127–146 (cit. on pp. [12](#), [179](#)).
- [Bos16] Raphael Bost. *Σοφός: Forward Secure Searchable Encryption*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1143–1154 (cit. on pp. [viii](#), [9](#), [10](#), [12](#), [56](#), [65](#)).
- [Bos17] Raphael Bost. *A State of the Art Single-Keyword Searchable Encryption Library in C/C++*. 2017. [Link](#). (Cit. on p. [88](#)).
- [BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. *Machine Learning Classification over Encrypted Data*. In: *NDSS 2015*. The Internet Society, Feb. 2015 (cit. on p. [xi](#)).
- [BR06] Mihir Bellare and Phillip Rogaway. *The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs*. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 409–426 (cit. on pp. [26](#), [31](#)).
- [BR93] Mihir Bellare and Phillip Rogaway. *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*. In: *ACM CCS 93*. Ed. by V. Ashby. ACM Press, Nov. 1993, pp. 62–73 (cit. on p. [27](#)).
- [BS13] Sumeet Bajaj and Radu Sion. *HIFS: history independence for file systems*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 1285–1296 (cit. on p. [115](#)).
- [BS16] Raphael Bost and Olivier Sanders. *Trick or Tweak: On the (In)security of OTR’s Tweaks*. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 333–353 (cit. on p. [xi](#)).
- [BW13] Dan Boneh and Brent Waters. *Constrained Pseudorandom Functions and Their Applications*. In: *ASIACRYPT 2013, Part II*. Ed. by Kazue Sako and Palash Sarkar. Vol. 8270. LNCS. Springer, Heidelberg, Dec. 2013, pp. 280–300 (cit. on p. [30](#)).
- [Can01] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. In: *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145 (cit. on p. [10](#)).
- [CCD+17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. *A Formal Security Analysis of the Signal Messaging Protocol*. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 451–466. ISBN: 978-1-5090-1751-5. [Link](#). (Cit. on p. [5](#)).

- [CDD+03] Dwaine E. Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. *Incremental Multiset Hash Functions and Their Application to Memory Integrity Checking*. In: *ASIACRYPT 2003*. Ed. by Chi-Sung Laih. Vol. 2894. LNCS. Springer, Heidelberg, Nov. 2003, pp. 188–207 (cit. on p. 33).
- [CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds*. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 3–33 (cit. on p. 6).
- [CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions*. In: *ACM CCS 06*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, Oct. 2006, pp. 79–88 (cit. on pp. 9, 42–45, 56, 72, 152).
- [CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. *Private Information Retrieval*. In: *36th FOCS*. IEEE Computer Society Press, Oct. 1995, pp. 41–50 (cit. on p. 5).
- [CGKS98] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. “Private Information Retrieval”. In: *Journal of the ACM* 45.6 (1998), pp. 965–982 (cit. on p. 5).
- [CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. *Leakage-Abuse Attacks Against Searchable Encryption*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 668–679 (cit. on pp. 60, 66, 151–153, 156, 161, 162, 170–172).
- [Cip] CipherCloud. *Cloud Data Encryption*. [Link](#). (Cit. on p. 10).
- [CJJ+13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 353–373 (cit. on pp. 9, 11, 43, 45, 58).
- [CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. *Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. 10, 49, 56, 73, 83, 99, 129, 163, 170, 171).
- [CK10] Melissa Chase and Seny Kamara. *Structured Encryption and Controlled Disclosure*. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 577–594 (cit. on pp. 9, 12, 43, 45, 57).
- [CL02] Jan Camenisch and Anna Lysyanskaya. *Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials*. In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 61–76 (cit. on p. 122).
- [CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 9780262033848 (cit. on p. 169).
- [CLWW16] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. *Practical Order-Revealing Encryption with Limited Leakage*. In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 474–493 (cit. on p. 11).

- [CM05] Yan-Cheng Chang and Michael Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*. In: *ACNS 05*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. LNCS. Springer, Heidelberg, June 2005, pp. 442–455 (cit. on pp. 9, 42).
- [Col15] Louis Columbus. *Roundup Of Small & Medium Business Cloud Computing Forecasts And Market Estimates, 2015*. May 2015. [Link](#). (Cit. on p. 4).
- [CT14] David Cash and Stefano Tessaro. *The Locality of Searchable Symmetric Encryption*. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 351–368 (cit. on pp. 57, 58, 68).
- [DDF+16] Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. *Onion ORAM: A Constant Bandwidth Blowup Oblivious RAM*. In: *TCC 2016-A, Part II*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9563. LNCS. Springer, Heidelberg, Jan. 2016, pp. 145–174 (cit. on pp. 7, 56).
- [DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption over the Integers*. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43 (cit. on p. 6).
- [DNRV09] Cynthia Dwork, Moni Naor, Guy N. Rothblum, and Vinod Vaikuntanathan. *How Efficient Can Memory Checking Be?* In: *TCC 2009*. Ed. by Omer Reingold. Vol. 5444. LNCS. Springer, Heidelberg, Mar. 2009, pp. 503–520 (cit. on p. 120).
- [Dou17] Doug Hoyte. *vmtouch - the Virtual Memory Toucher*. 2017. [Link](#). (Cit. on p. 91).
- [DP17] Ioannis Demertzis and Charalampos Papamanthou. *Fast Searchable Encryption With Tunable Locality*. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM. 2017, pp. 1053–1067 (cit. on pp. 58, 59, 68, 69).
- [ElG84] Taher ElGamal. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 10–18 (cit. on p. 6).
- [Enron] Enron Corporation. *Enron email dataset*. [Link](#). (Cit. on pp. 170, 172).
- [EU16] *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. European Union, Apr. 2016. [Link](#). (Cit. on p. 4).
- [EU95] *DIRECTIVE 95/46/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data*. European Union, Oct. 1995. [Link](#). (Cit. on p. 4).
- [Fac17] Facebook, Inc. *RocksDB: A Persistent Key-Value Store for Fast Storage Environments*. Sept. 2017. [Link](#). (Cit. on p. 89).
- [FGK17] Xiong Fan, Chaya Ganesh, and Vladimir Kolesnikov. *Hashing Garbled Circuits for Free*. In: *EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. LNCS. Springer, Heidelberg, May 2017, pp. 456–485 (cit. on p. 7).

- [FJK+15] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. *Rich Queries on Encrypted Data: Beyond Exact Matches*. In: *ESORICS 2015, Part II*. Ed. by Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl. Vol. 9327. LNCS. Springer, Heidelberg, Sept. 2015, pp. 123–145 (cit. on p. 11).
- [FVY+17] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. *SoK: Cryptographically Protected Database Search*. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 172–191 (cit. on p. 12).
- [Gas04] William Gasarch. “A survey on private information retrieval”. In: *The Bulletin of the EATCS* 82.72-107 (2004), p. 1 (cit. on p. 6).
- [Gen09] Craig Gentry. *Fully homomorphic encryption using ideal lattices*. In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178 (cit. on p. 6).
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. *How to Construct Random Functions (Extended Abstract)*. In: *25th FOCS*. IEEE Computer Society Press, Oct. 1984, pp. 464–479 (cit. on pp. 82, 87).
- [GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. *Protecting Data Privacy in Private Information Retrieval Schemes*. In: *30th ACM STOC*. ACM Press, May 1998, pp. 151–160 (cit. on p. 6).
- [GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. *Reusable garbled circuits and succinct functional encryption*. In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 555–564 (cit. on p. 7).
- [GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. *Fast Garbling of Circuits Under Standard Assumptions*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 567–578 (cit. on p. 7).
- [GM11] Michael T. Goodrich and Michael Mitzenmacher. *Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation*. In: *ICALP 2011, Part II*. Ed. by Luca Aceto, Monika Henzinger, and Jiri Sgall. Vol. 6756. LNCS. Springer, Heidelberg, July 2011, pp. 576–587 (cit. on pp. 59, 71, 130, 133, 147).
- [GM15] Matthew D. Green and Ian Miers. *Forward Secure Asynchronous Messaging from Puncturable Encryption*. In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 305–320 (cit. on pp. 104, 106).
- [GM16] John F. Gantz and Pam Miller. *The Salesforce Economy: Enabling 1.9 Million New Jobs and \$389 Billion in New Revenue Over the Next Five Years*. Sept. 2016. [Link](#). (Cit. on p. 4).
- [GM84] Shafi Goldwasser and Silvio Micali. “Probabilistic Encryption”. In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 270–299 (cit. on p. 6).
- [GMOT12] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. *Privacy-preserving group data access via stateless oblivious RAM simulation*. In: *23rd SODA*. Ed. by Yuval Rabani. ACM-SIAM, Jan. 2012, pp. 157–167 (cit. on pp. 59, 71).

- [GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. *TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption*. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563–592 (cit. on pp. 7, 8, 45, 69, 73, 99, 101, 161, 163, 170).
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. *How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority*. In: *19th ACM STOC*. Ed. by Alfred Aho. ACM Press, May 1987, pp. 218–229 (cit. on p. 7).
- [GO96] Oded Goldreich and Rafail Ostrovsky. “Software protection and simulation on oblivious RAMs”. In: *Journal of the ACM* 43.3 (1996), pp. 431–473 (cit. on pp. 7, 8, 51, 52, 56, 82).
- [Goh03] Eu-Jin Goh. *Secure Indexes*. Cryptology ePrint Archive, Report 2003/216. <http://eprint.iacr.org/2003/216>. 2003 (cit. on pp. 9, 11, 42).
- [Gol04] Oded Goldreich. *Foundations of cryptography*. Cambridge University Press, 2004 (cit. on pp. 29, 44, 47).
- [Goo17] Google, Inc. *gRPC: A high performance, open-source universal RPC framework*. Sept. 2017. [Link](#). (Cit. on p. 89).
- [GPS06] S.D. Galbraith, K.G. Paterson, and N.P. Smart. *Pairings for Cryptographers*. Cryptology ePrint Archive, Report 2006/165. <http://eprint.iacr.org/2006/165>. 2006 (cit. on pp. 29, 115).
- [Gre14] Glenn Greenwald. *No place to hide: Edward Snowden, the NSA, and the US surveillance state*. Macmillan, 2014 (cit. on p. 4).
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 75–92 (cit. on p. 6).
- [HR18] Godfrey H. Hardy and Srinivasa Ramanujan. “Asymptotic formulæ in combinatory analysis”. In: *Proceedings of the London Mathematical Society* 2.1 (1918), pp. 75–115 (cit. on p. 168).
- [IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. *Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation*. In: *NDSS 2012*. The Internet Society, Feb. 2012 (cit. on pp. 59, 60, 151, 152, 173).
- [Int] Intel. *Intel®SSD 750 Series: Performance Unleashed*. [Link](#). (Cit. on p. 91).
- [Int17] Intel. *Intel®Software Guard Extensions: An Intel®architecture extension designed to increase the security of application code and data*. July 2017. [Link](#). (Cit. on p. 179).
- [KA16] Jan Koum and Brian Acton. *WhatsApp Blog: end-to-end encryption*. Apr. 2016. [Link](#). (Cit. on p. 5).
- [Ker15] Florian Kerschbaum. *Frequency-Hiding Order-Preserving Encryption*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 656–667 (cit. on p. 11).

- [KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O’Neill. *Generic Attacks on Secure Outsourced Databases*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1329–1340 (cit. on pp. 151–153, 160).
- [KLO12] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. *On the (in)security of hash-based oblivious RAM and a new balancing scheme*. In: *23rd SODA*. Ed. by Yuval Rabani. ACM-SIAM, Jan. 2012, pp. 143–156 (cit. on p. 82).
- [KM15] Neal Koblitz and Alfred Menezes. *The Random Oracle Model: A Twenty-Year Retrospective*. Cryptology ePrint Archive, Report 2015/140. <http://eprint.iacr.org/2015/140>. 2015 (cit. on p. 27).
- [KM16] Seny Kamara and Tarik Moataz. *SQL on Structurally-Encrypted Databases*. Cryptology ePrint Archive, Report 2016/453. <http://eprint.iacr.org/2016/453>. 2016 (cit. on p. 12).
- [KM17] Seny Kamara and Tarik Moataz. *Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity*. In: *EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. LNCS. Springer, Heidelberg, May 2017, pp. 94–124 (cit. on pp. 9, 11).
- [KO12] Kaoru Kurosawa and Yasuhiro Ohtaki. *UC-Secure Searchable Symmetric Encryption*. In: *FC 2012*. Ed. by Angelos D. Keromytis. Vol. 7397. LNCS. Springer, Heidelberg, Feb. 2012, pp. 285–298 (cit. on pp. 9, 10, 47).
- [KO13] Kaoru Kurosawa and Yasuhiro Ohtaki. *How to Update Documents Verifiably in Searchable Symmetric Encryption*. In: *CANS 13*. Ed. by Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab. Vol. 8257. LNCS. Springer, Heidelberg, Nov. 2013, pp. 309–328 (cit. on pp. 10, 47).
- [KO97] Eyal Kushilevitz and Rafail Ostrovsky. *Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval*. In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 364–373 (cit. on pp. 6, 153).
- [KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. *Dynamic searchable symmetric encryption*. In: *ACM CCS 12*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. ACM Press, Oct. 2012, pp. 965–976 (cit. on pp. 9, 10).
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. *Delegatable pseudorandom functions and applications*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 669–684 (cit. on pp. 30, 87).
- [KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. *On the Security of the TLS Protocol: A Systematic Analysis*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 429–448 (cit. on p. 5).
- [KS08] Vladimir Kolesnikov and Thomas Schneider. *Improved Garbled Circuit: Free XOR Gates and Applications*. In: *ICALP 2008, Part II*. Ed. by Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz. Vol. 5126. LNCS. Springer, Heidelberg, July 2008, pp. 486–498 (cit. on p. 7).



- [Lip10] Helger Lipmaa. *First CPIR Protocol with Data-Dependent Computation*. In: *ICISC 09*. Ed. by Donghoon Lee and Seokhie Hong. Vol. 5984. LNCS. Springer, Heidelberg, Dec. 2010, pp. 193–210 (cit. on p. 6).
- [LMP17] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. *Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage*. Cryptology ePrint Archive, Report 2017/701. <http://eprint.iacr.org/2017/701>. 2017 (cit. on p. 12).
- [LO13a] Steve Lu and Rafail Ostrovsky. *Distributed Oblivious RAM for Secure Two-Party Computation*. In: *TCC 2013*. Ed. by Amit Sahai. Vol. 7785. LNCS. Springer, Heidelberg, Mar. 2013, pp. 377–396 (cit. on p. 7).
- [LO13b] Steve Lu and Rafail Ostrovsky. *How to Garble RAM Programs*. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 719–734 (cit. on pp. 8, 56).
- [Mie15] Ian Miers. *Libforwardsec. Forward secure encryption for asynchronous messaging*. 2015. [Link](#). (Cit. on p. 115).
- [Min14] Kazuhiko Minematsu. *Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions*. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 275–292 (cit. on p. xi).
- [MKNK15] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. *GRECS: Graph Encryption for Approximate Shortest Distance Queries*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 504–517 (cit. on p. 12).
- [MM17] Ian Miers and Payman Mohassel. *IO-DSSE: Scaling Dynamic Searchable Encryption to Millions of Indexes By Improving Locality*. In: *NDSS 2017*. The Internet Society, 2017 (cit. on p. 82).
- [MMBC15] Tarik Moataz, Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. *Resizable Tree-Based Oblivious RAM*. In: *FC 2015*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. LNCS. Springer, Heidelberg, Jan. 2015, pp. 147–167 (cit. on p. 68).
- [MP16] Moxie Marlinspike and Trevor Perrin. *The Double Ratchet Algorithm*. 2016. [Link](#). (Cit. on p. 5).
- [MTA16] Jeremy Maitin-Shepard, Mehdi Tibouchi, and Diego F Aranha. “Elliptic Curve Multiset Hash”. In: *The Computer Journal* 60.4 (2016), pp. 476–490 (cit. on pp. 33, 129).
- [MZK15] Xianrui Meng, Haohan Zhu, and George Kollios. *Top-k Query Processing on Encrypted Databases with Strong Security Guarantees*. arXiv preprint arXiv:1510.05175. 2015 (cit. on p. 12).
- [NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. *Inference Attacks on Property-Preserving Encrypted Databases*. In: *ACM CCS 15*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 644–655 (cit. on pp. 10, 12, 59).
- [NP99] Moni Naor and Benny Pinkas. *Oblivious Transfer with Adaptive Queries*. In: *CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 573–590 (cit. on p. 6).
- [NR05] Moni Naor and Guy N. Rothblum. *The Complexity of Online Memory Checking*. In: *46th FOCS*. IEEE Computer Society Press, Oct. 2005, pp. 573–584 (cit. on p. 120).

- [NT01] Moni Naor and Vanessa Teague. *Anti-persistence: History independent data structures*. In: *33rd ACM STOC*. ACM Press, July 2001, pp. 492–501 (cit. on p. 114).
- [OS07] Rafail Ostrovsky and William E. Skeith III. *A Survey of Single-Database Private Information Retrieval: Techniques and Applications (Invited Talk)*. In: *PKC 2007*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Vol. 4450. LNCS. Springer, Heidelberg, Apr. 2007, pp. 393–411 (cit. on p. 6).
- [Pai99] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238 (cit. on p. 6).
- [PBP16] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. *Arx: A Strongly Encrypted Database System*. Cryptology ePrint Archive, Report 2016/591. <http://eprint.iacr.org/2016/591>. 2016 (cit. on pp. 9, 12, 87, 99).
- [PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. *Hash Functions Based on Block Ciphers: A Synthetic Approach*. In: *CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 368–378 (cit. on p. 88).
- [PKV+14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. *Blind Seer: A Scalable Private DBMS*. In: *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 359–374 (cit. on p. 11).
- [PLZ13] Raluca A. Popa, Frank H. Li, and Nikolai Zeldovich. *An Ideal-Security Protocol for Order-Preserving Encoding*. In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 463–477 (cit. on p. 11).
- [PRZB11] Raluca Ada Popa, Catherine Redfield, Nikolai Zeldovich, and Hari Balakrishnan. *CryptDB: protecting confidentiality with encrypted query processing*. In: *ACM SOSP 11*. ACM, 2011, pp. 85–100 (cit. on p. 11).
- [PT08] Charalampos Papamanthou and Roberto Tamassia. *Time and Space Efficient Algorithms for Two-Party Authenticated Data Structures*. In: *ICICS 07*. Ed. by Sihon Qing, Hideki Imai, and Guilin Wang. Vol. 4861. LNCS. Springer, Heidelberg, Dec. 2008, pp. 1–15 (cit. on p. 120).
- [PTT09] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. *Cryptographic Accumulators for Authenticated Hash Tables*. Cryptology ePrint Archive, Report 2009/625. <http://eprint.iacr.org/2009/625>. 2009 (cit. on pp. 120, 122, 129).
- [Pub01] NIST FIPS Pub. “197: Advanced encryption standard (AES)”. In: *Federal information processing standards publication 197.441* (Nov. 2001), p. 0311. [Link](#). (Cit. on p. 88).
- [RACY16] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. *POPE: Partial Order Preserving Encoding*. In: *ACM CCS 16*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1131–1142 (cit. on p. 11).
- [RAD78] Ronald Rivest, Leonard Adleman, and Michael Dertouzos. “On data banks and privacy homomorphisms”. In: *Foundations of secure computation* 4.11 (1978), pp. 169–180 (cit. on p. 6).

- [RBB03] Phillip Rogaway, Mihir Bellare, and John Black. “OCB: A block-cipher mode of operation for efficient authenticated encryption”. In: *ACM Transactions on Information and System Security (TISSEC)* 6.3 (2003), pp. 365–403 (cit. on p. 133).
- [RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on pp. 6, 27).
- [SA15] M-J. Saarinen and J-P. Aumasson. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. RFC 7693. RFC Editor, Nov. 2015 (cit. on p. 88).
- [Sam16] Samsung. *Samsung SSD 850 EVO Data Sheet, Rev.3.1*. May 2016. [Link](#). (Cit. on p. 91).
- [SDS+13a] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. *Path ORAM: an extremely simple oblivious RAM protocol*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 299–310 (cit. on pp. 7, 56, 82).
- [SDS+13b] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. *Path ORAM: An Extremely Simple Oblivious RAM Protocol*. Cryptology ePrint Archive, Report 2013/280. <http://eprint.iacr.org/2013/280>. 2013 (cit. on p. 82).
- [sky] skyhigh. *Cloud Security and Enablement*. [Link](#). (Cit. on p. 10).
- [SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. In: *NDSS 2014*. The Internet Society, Feb. 2014 (cit. on pp. viii–x, 10, 13, 45, 59, 67, 71, 73, 81, 97–99, 115, 119, 122, 130–133, 141, 147, 163, 170).
- [SS13] Emil Stefanov and Elaine Shi. *Multi-cloud oblivious storage*. In: *ACM CCS 13*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 247–258 (cit. on p. 7).
- [STY01] Tomas Sander, Amnon Ta-Shma, and Moti Yung. *Blind, Auditable Membership Proofs*. In: *FC 2000*. Ed. by Yair Frankel. Vol. 1962. LNCS. Springer, Heidelberg, Feb. 2001, pp. 53–71 (cit. on p. 122).
- [SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. *Practical Techniques for Searches on Encrypted Data*. In: *2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2000, pp. 44–55 (cit. on pp. 8, 9, 42).
- [TT05] Roberto Tamassia and Nikos Triandopoulos. *Computational Bounds on Hierarchical Data Processing with Applications to Information Security*. In: *ICALP 2005*. Ed. by Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung. Vol. 3580. LNCS. Springer, Heidelberg, July 2005, pp. 153–165 (cit. on pp. 120, 122, 129).
- [USA02] *Sarbanes-Oxley Act of 2002*. United States of America, July 2002. [Link](#). (Cit. on p. 4).
- [USA96] *Health Insurance Portability and Accountability Act of 1996*. United States of America, Aug. 1996. [Link](#). (Cit. on p. 4).
- [Vau02] Serge Vaudenay. *Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS...* In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, Apr. 2002, pp. 534–546 (cit. on p. 10).

- [WHC+14] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. *SCORAM: Oblivious RAM for Secure Computation*. In: *ACM CCS 14*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM Press, Nov. 2014, pp. 191–202 (cit. on p. 8).
- [WNL+14] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. *Oblivious Data Structures*. In: *ACM CCS 14*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM Press, Nov. 2014, pp. 215–226 (cit. on p. 8).
- [WS12] Peter Williams and Radu Sion. *Single round access privacy on outsourced storage*. In: *ACM CCS 12*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. ACM Press, Oct. 2012, pp. 293–304 (cit. on p. 7).
- [Yao82] Andrew Chi-Chih Yao. *Protocols for Secure Computations (Extended Abstract)*. In: *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164 (cit. on p. 7).
- [Yao86] Andrew Chi-Chih Yao. *How to Generate and Exchange Secrets (Extended Abstract)*. In: *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 162–167 (cit. on p. 7).
- [ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. *All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption*. In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 2016, pp. 707–720 (cit. on pp. 10, 12, 60, 66, 76, 114, 152, 153, 156, 157).

# List of Figures

2.1	Constrained PRF security game . . . . .	30
2.2	EUUF-CMA security game . . . . .	35
3.1	SSECORR $_{\Sigma}$ : SSE correctness game . . . . .	42
3.2	SSEIND $_{\Sigma, \mathcal{L}}$ : SSE indistinguishability game . . . . .	43
3.3	SSEREAL and SSEIDEAL: simulation-based security games . . . . .	45
3.4	SSE indistinguishability game against malicious adversaries . . . . .	46
3.5	SSE simulation-based security games against malicious adversaries . . . . .	46
3.6	SSESOUND $_{\Sigma}$ : SSE soundness game . . . . .	47
4.1	Relations among search and update tokens in $\Sigma\phi\phi\phi\phi\phi$ . . . . .	74
4.2	Token generation in Diana . . . . .	88
4.3	Software architecture of OpenSSE . . . . .	89
4.4	Search performance of $\Sigma\phi\phi\phi\phi\phi$ and Diana . . . . .	90
5.1	Puncturable encryption security game . . . . .	105
6.1	Reduction from memory checkers to verifiable SSE . . . . .	121
6.2	Correctness and soundness games for verifiable hash tables . . . . .	124
6.3	Non-consecutive hole attack against SPS . . . . .	132
6.4	Example of what CheckResults checks . . . . .	141
7.1	SSEIND $_{\Sigma, \mathcal{L}, \mathcal{C}}$ : SSE constrained indistinguishability game . . . . .	155
7.2	SSEIND $_{\Sigma, \mathcal{L}, \mathcal{C}, k}$ : Extended SSE constrained indistinguishability game . . . . .	158
7.3	SSEIND $_{\Sigma, \mathcal{L}, \mathcal{D}}$ : SSE distribution indistinguishability game . . . . .	160
7.4	Example of padding overhead . . . . .	171
7.5	Running time of the count attack in presence of padding . . . . .	172



# List of Tables

4.1	Comparison of dynamic SSE schemes with respect to forward privacy . . . . .	73
4.2	Size of the evaluation databases. . . . .	89
5.1	Comparison of different dynamic SSE schemes with respect to backward privacy . .	99
5.2	Performance of the puncturable encryption scheme used in Janus . . . . .	115
6.1	Computational complexity of GVSSE depending on the VHT instantiation used . .	129
7.1	Running time of the count attack with two types of padding . . . . .	172





# List of Algorithms

4.1	Forward privacy from static SSE. $\Sigma$ is a static SSE scheme. . . . .	72
4.2	$\Sigma\phi\phi\phi\phi$ -B: Forward private SSE scheme with large client storage. . . . .	75
4.3	Proof of $\Sigma\phi\phi\phi\phi$ : games $G_2$ and $\widetilde{G}_2$ . . . . .	77
4.4	Proof of $\Sigma\phi\phi\phi\phi$ : game $G_4$ . . . . .	79
4.5	Proof of $\Sigma\phi\phi\phi\phi$ : simulator $S$ . . . . .	80
4.6	FS-RCPRF: Forward private SSE scheme from range-constrained PRF . . . . .	83
4.7	Proof of FS-RCPRF: games $G_2$ and $\widetilde{G}_2$ . . . . .	85
4.8	Proof of FS-RCPRF: game $G_4$ . . . . .	86
5.9	Generic backward-private scheme $B(\Sigma)$ . . . . .	101
5.10	Improved generic backward-private scheme $B(\Sigma)$ . . . . .	102
5.11	Green-Miers puncturable encryption . . . . .	106
5.12	Janus: weakly backward-secure SSE. . . . .	109
5.13	Proof of Janus: game $G_2$ . . . . .	110
5.14	Proof of Janus: game $G_3$ . . . . .	111
5.15	Proof of Janus: game $G_4$ . . . . .	112
5.16	Adaptation of Green-Miers' scheme for pseudo-random generation of parameters and randomness . . . . .	114
6.17	Instantiation of a static verifiable hash table . . . . .	125
6.18	The GVSSE construction . . . . .	128
6.19	Verif-SPS construction . . . . .	133
6.20	Verif-SPS: modified Rebuild protocol . . . . .	134
6.21	Verif-SPS: Simple Search algorithm . . . . .	135
6.22	Verif-SPS: Sublinear Search algorithm . . . . .	136
6.23	Verif-SPS: ProcessLevel algorithm . . . . .	137
6.24	Verif-SPS: Prove and verify holes . . . . .	139
6.25	Verif-SPS: CheckResults algorithm . . . . .	140





## **Résumé**

La recherche sur les bases de données chiffrées vise à rendre efficace une tâche apparemment simple : déléguer le stockage de données à un serveur qui ne serait pas de confiance, tout en conservant des fonctionnalités de recherche. Avec le développement des services de stockage dans le Cloud, destinés aussi bien aux entreprises qu'aux individus, la mise au point de solutions efficaces à ce problème est essentielle pour permettre leur déploiement à large échelle. Le principal problème de la recherche sur bases de données chiffrées est qu'un schéma avec une sécurité parfaite implique un surcoût en termes de calcul et de communication qui serait inacceptable pour des fournisseurs de services sur le Cloud ou pour les utilisateurs — tout du moins avec les technologies actuelles.

Cette thèse propose et étudie de nouvelles notions de sécurité et de nouvelles constructions de bases de données chiffrées permettant des recherches efficaces et sûres. En particulier, nous considérons la confidentialité persistante et la confidentialité future de ces bases de données, ce que ces notions impliquent en termes de sécurité et d'efficacité, et comment les réaliser. Ensuite, nous montrons comment protéger les utilisateurs de bases de données chiffrées contre des attaques actives de la part du serveur hébergeant la base, et que ces protections ont un coût inévitable. Enfin, nous étudions les attaques existantes contre ces bases de données chiffrées et comment les éviter.

## **Abstract**

Searchable encryption aims at making efficient a seemingly easy task: outsourcing the storage of a database to an untrusted server, while keeping search features. With the development of Cloud storage services, for both private individuals and businesses, efficiency of searchable encryption became crucial: inefficient constructions would not be deployed on a large scale because they would not be usable. The key problem with searchable encryption is that any construction achieving 'perfect security' induces a computational or a communication overhead that is unacceptable for the providers or for the users — at least with current techniques and by today's standards.

This thesis proposes and studies new security notions and new constructions of searchable encryption, aiming improving efficiency and security. In particular, we start by considering the forward and backward privacy of searchable encryption schemes, what it implies in terms of security and efficiency, and how we can realize them. Then, we show how to protect an encrypted database user against active attacks by the Cloud provider, and that such protections have an inherent efficiency cost. Finally, we take a look at existing attacks against searchable encryption, and explain how we might thwart them.