ОБРПО. Лекция 9 Криптография на практике

ТЮРИН КАЙ АНДРЕЕВИЧ

О чём сегодняшняя лекция

Об основных понятиях криптографии и о том, что использовать на практике.

В этой лекции нет теоретических основ криптографии.

Хеши

Что такое хеш-функция?

Хеширование — преобразование массива входных данных произвольной длины в битовую последовательность фиксированной длины

Хеш-функций используются для:

- ассоциативных массивов;
- поиска одинаковых данных;
- построения уникальных идентификаторов;
- обнаружения ошибок при передаче/хранении данных;
- при сохранении паролей;
- при создании электронной подписи.

Требования к хешам

К криптографическим хеш-функциям предъявляются следующие требования:

1. Стойкость к поиску первого прообраза: при известном H(m) тяжело найти m.

2. Стойкость к поиску второго прообраза: при известных H(m) и m, тяжело найти m′ ≠ m , чтобы H(m) = H(m′).

3. Стойкость к коллизиям: эффективного полиномиального алгоритма, позволяющего находить коллизии (m и m′, такие, что m′ ≠ m и H(m) = H(m′)).

Также является важным, чтобы значения хеш-функции сильно изменялись при малейшем изменении аргумента (лавинный эффект).

Свойства не являются независимыми

Обратимая функция неустойчива к восстановлению второго прообраза и коллизиям.

Функция, нестойкая к восстановлению второго прообраза, нестойка к коллизиям; обратное неверно.

Функция устойчивая к коллизиям, устойчива к нахождению второго прообраза.

Устойчивая к коллизиям хеш-функция не обязательно является односторонней.

Требования к ГОСТ Р 34.11-2012 (Стрибог)

Сложность к вычислению прообраза: если известно значение функции, тогда должно быть сложно найти такое сообщение, хеш-функция от которого равна известному;

Стойкость вычисления второго прообраза: пусть есть одно значение, и известен хеш-код этого значения. Тогда злоумышленнику должно быть сложно найти еще одно такое значение, чтобы его хеш-функция совпадала с хеш-функцией первого значения;

Сложность к поиску коллизий: должно быть сложно найти два таких сообщения, которые не равны, но у них равны хеш-коды;

Стойкость к удлинению прообраза: если злоумышленник не знает сообщение, но знает его длину и хеш-код от него, то ему должно быть сложно подобрать такое сообщение, которое, будучи дописанным к оригинальному, даст какую-нибудь известную хеш-функцию. Другими словами, не должно быть возможно злоумышленнику что-то менять путем дополнения в сообщении, получая известный выход.

CRC

Циклический избыточный код (Cyclic redundancy check).

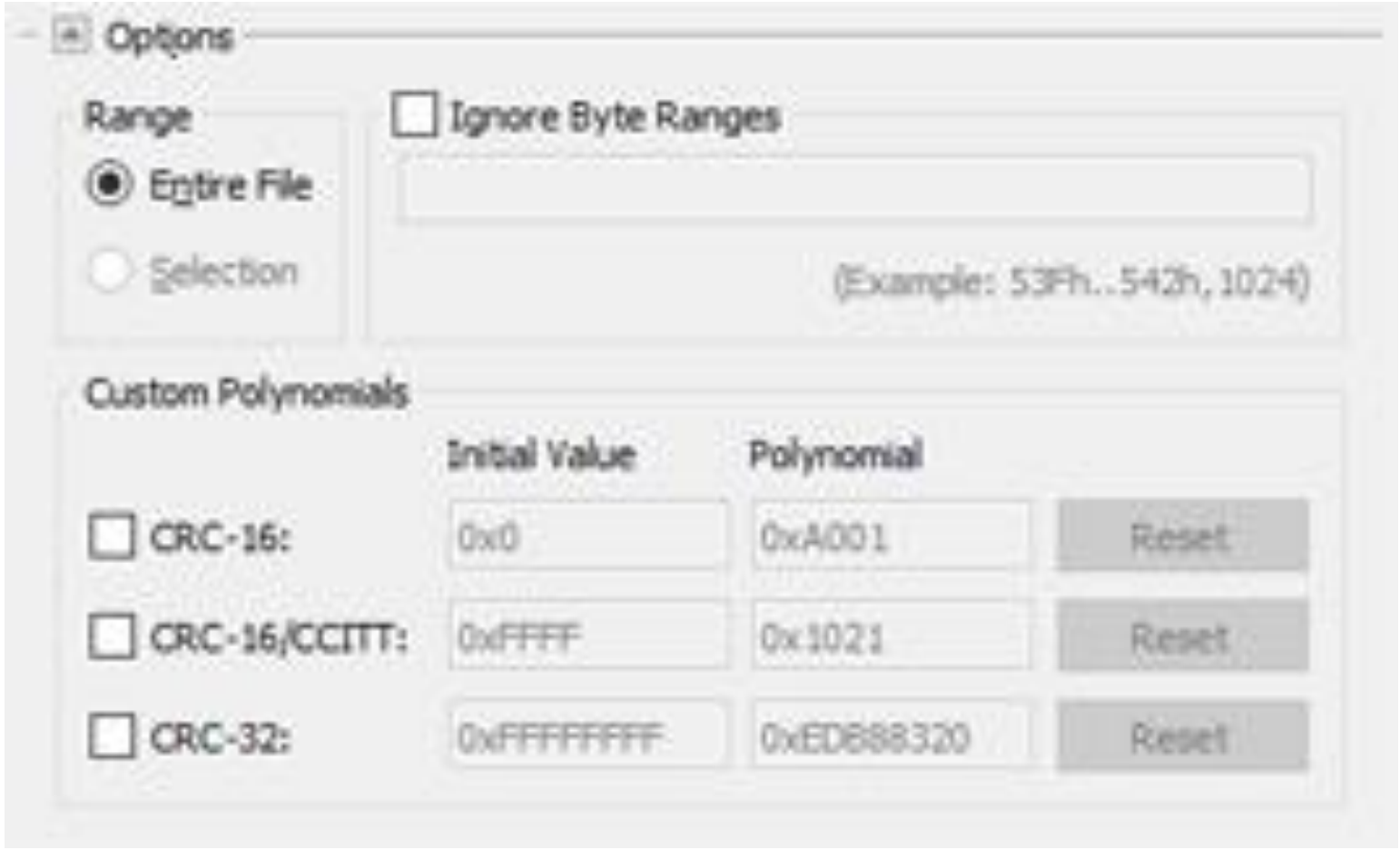
Основан на операциях над полиномами.

НЕ является криптографическим и подходит только для проверки целостности.

Обычно говорят о CRC-16, CRC-32.

Основан на порождающем полиноме, который должен совпадать для разных вычислений, если мы хотим иметь возможность сравнивать CRC-коды.

CRC


Options
Range
Entire File
Selection
Ignore Byte Ranges
(Example: 53Fh..542h,1024)
Custom Polynomials
Initial Value
Polynomial
CRC-16:
0x0
0xA001
Reset
CRC-16/CCITT:
0xFFFF
0x1021
Reset
CRC-32:
0xFFFFFFFF
0xEDB88320
Reset

Message Digest: MD2, MD4

MD2 — криптографическая хеш-функция, разработанная Рональдом Ривестом в 1989 году, и описанная в RFC 1319. На входе сообщение произвольный длины. Размер хеша — 128 бит. Роже и Шаво в 1997 году опубликовали пример коллизий для MD2. Первую атаку на MD2 целиком в 2004 году предложил Фредерих Мюллер, позволяющую найти прообраз с трудоёмкостью 2^{104} операций.

MD4 — криптографическая хеш-функция, разработанная профессором Массачусетского университета Рональдом Ривестом в 1990 году, и впервые описанная в RFC 1186. Для произвольного входного сообщения функция генерирует 128-разрядное хеш-значение, называемое дайджестом сообщения. Уязвимости в MD4 были продемонстрированы в статье Берта ден Бура и Антона Босселарса в 1991 году. Первая коллизия была найдена Гансом Доббертином в 1996 году.

MD5, MD6

Алгоритм MD5 имел некогда большую популярность, но первые предпосылки взлома появились еще в конце девяностых, и сейчас его популярность стремительно падает. В 2004 году китайские исследователи Ван Сяоюнь (Wang Xiaoyun), Фэн Дэнго (Feng Dengguo), Лай Сюэцзя (Lai Xuejia) и Юй Хунбо (Yu Hongbo) объявили об обнаруженной ими уязвимости в алгоритме, позволяющей за небольшое время (1 час на кластере IBM p690) находить коллизии. В 2006 году чешский исследователь Властимил Клима опубликовал алгоритм, позволяющий находить коллизии на обычном компьютере с любым начальным вектором (A,B,C,D)

Алгоритм MD6 — выдвигался на конкурс SHA-3, но, не был доработан и не прошёл во второй раунд.

Ещё раз: MD5 НЕ ЯВЛЯЕТСЯ БЕЗОПАСНОЙ ХЕШ-ФУНКЦИЕЙ

SHA-1

Описан в RFC 3174. Для входного сообщения произвольной длины (максимум 22^{64}-1 бит, что примерно равно 2 эксабайта) алгоритм генерирует 160-битное хеш-значение.

23 февраля 2017 года специалисты из Google и CWI объявили о практическом взломе алгоритма, опубликовав 2 PDF-файла с одинаковой контрольной суммой SHA-1. Это потребовало перебора $9* 10^{18}$ вариантов (110 лет на 1 GPU).

SHA-2

Семейство криптографических алгоритмов, включающее в себя алгоритмы SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/256 и SHA-512/224.

На данный момент является стандартом хеширования.

Bitcoin использует SHA256.

Тем не менее, в виду алгоритмической схожести SHA-2 с SHA-1 и наличия у последней потенциальных уязвимостей принято решение, что SHA-3 будет базироваться на совершенно ином алгоритме.

SHA-3

2 октября 2012 года NIST утвердил в качестве SHA-3 алгоритм Keccak.

Алгоритм хеширования переменной разрядности, разработанный группой авторов во главе с Йоаном Дайменом, соавтором Rijndael, автором шифров MMB, SHARK, Noekeon, SQUARE и BaseKing.

На данный момент не является широко распространённым.

ГОСТ Р 34.11-94

Дата введения: 23 мая 1994 года

Дата отмены: 1 января 2013 года.

Размер хеша: 256 бит

Размер блока входных данных: 256 бит

Разработчик: ГУБС ФАПСИ и Всероссийский научно-исследовательский институт стандартизации

В 2008 году командой экспертов из Австрии и Польши была обнаружена техническая уязвимость, сокращающая поиск коллизий в 2^{23} раз. Количество операций, необходимое для нахождения коллизии, таким образом, составляет 2^{105}.

ГОСТ Р 34.11-2012 Стрибог

Разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС» и введен в действие 1 января 2013 года.

Размер хеша — 256 или 512 бит.

Некоторые исследователи предполагают наличие бэкдора в алгоритме

Шифрование

Шифрование

Шифрование — обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения конфиденциальности передаваемой информации.

Алгоритмы шифрования делятся на

- Симметричные – используется 1 секретный ключ, которым обладают оба собеседника
- Ассиметричные – используются пары открытый/закрытый ключ. Шифрование происходит посредством открытого ключа, дешифрование – закрытым.

Симметричные криптоалгоритмы

XOR

Или «гаммирование» на русском.

Побайтовый XOR с гаммой, циклически наложенной на входные данные.

В случае, если количество нулей и единиц в гамме одинаково, а её длина равна длине сообщения, гаммирование является абсолютно стойким.

В любом другом случае это очень слабый способ шифрования.

DES

DES (англ. Data Encryption Standard) — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт. Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.

Прямым развитием DES в настоящее время является алгоритм Triple DES (3DES). В 3DES шифрование/расшифровка выполняются путём троекратного выполнения алгоритма DES.

Не рекомендуется для использования.

AES (Rijndael)

Симметричный алгоритм блочного шифрования. Размер одного блока. 128 бит, ключи 128/192/256, принят стандартом правительством США по результатам конкурса AES.

Пришел на смену алгоритму DES. Спецификация была опубликована 26 ноября 2001 год. 26 мая 2002 был объявлен стандартом шифрования. До сих пор является одним из самых распространенных алгоритмов симметричного шифрования.

Salsa20 и Chacha20

Salsa20 — система поточного шифрования, разработанная Даниэлем Бернштейном. Алгоритм был представлен на конкурсе «eSTREAM», целью которого было создание европейских стандартов для шифрования данных, передаваемых почтовыми системами. Алгоритм стал победителем конкурса в первом профиле (поточные шифры для программного применения с большой пропускной способностью).

Шифр Salsa20 использует следующие операции:

- сложение 32-битных чисел;
- побитовое сложение по модулю 2 (xor);
- сдвиги битов.

Chacha20 является модификацией Salsa20, но предоставляет большую криптостойкость и работает быстрее.

Магма

ГОСТ 28147-89 (Магма) — российский стандарт симметричного блочного шифрования, принятый в 1989 году. Полное название — «ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования». Является примером DES-подобных криптосистем, созданных по классической итерационной схеме Фейстеля.

Кузнечик

Блочный шифр «Кузнечик» (входит в стандарт ГОСТ Р 34.12-2015) — симметричный алгоритм блочного шифрования с размером блока 128 бит и длиной ключа 256 бит и использующий для генерации раундовых ключей сеть Фейстеля.

Данный шифр утверждён (наряду с блочным шифром «Магма») в качестве стандарта в ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры» приказом от 19 июня 2015 года № 749-ст. Стандарт вступил в действие с 1 января 2016 года. Шифр разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «Информационные технологии и коммуникационные системы» (ОАО «ИнфоТеКС»).

Ассиметричные криптоалгоритмы

RSA

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Система RSA используется для защиты программного обеспечения и в схемах цифровой подписи.

Также она используется в открытой системе шифрования PGP и иных системах шифрования (к примеру, DarkCryptTC и формат xdc) в сочетании с симметричными алгоритмами.

Ускорение шифрования

Из-за низкой скорости шифрования (около 30 кбит/с при 512-битном ключе на процессоре 2 ГГц), сообщения обычно шифруют с помощью более производительных симметричных алгоритмов со случайным сеансовым ключом (например, AES, IDEA, Serpent, Twofish), а с помощью RSA шифруют лишь этот ключ, таким образом реализуется гибридная криптосистема. Такой механизм имеет потенциальные уязвимости ввиду необходимости использовать криптографически стойкий генератор псевдослучайных чисел для формирования случайного сеансового ключа симметричного шифрования.

Схема Эль-Гамаля

Схема Эль-Гамаля (Elgamal) — криптосистема с открытым ключом, основанная на трудности вычисления дискретных логарифмов в конечном поле. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Схема Эль-Гамаля лежит в основе бывших стандартов электронной цифровой подписи в США (DSA) и России (ГОСТ Р 34.10-94).

Основана на проблеме дискретного логарифмирования.

Протокол Диффи-Хеллмана

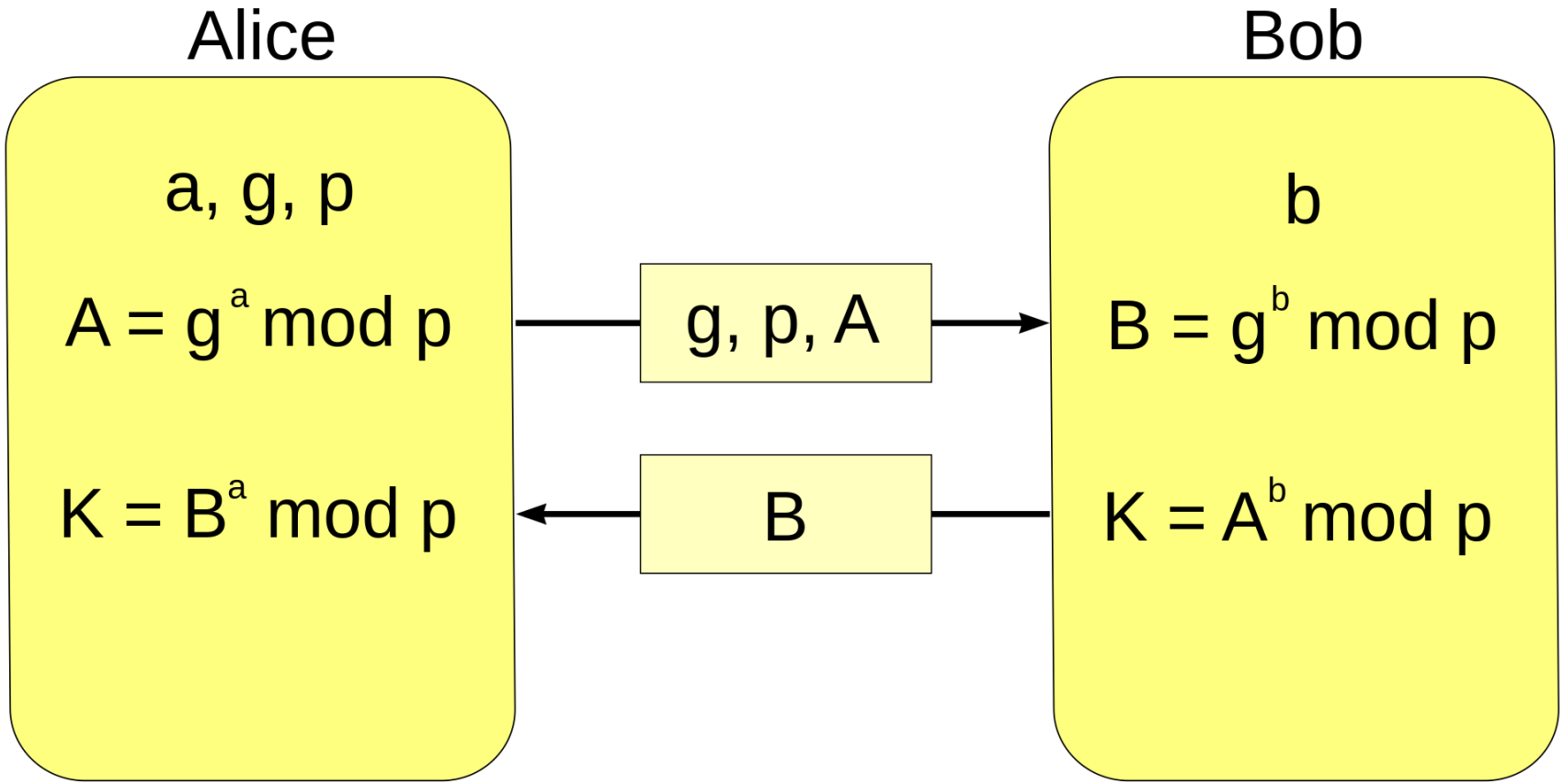
Ассиметричный протокол генерации симметричного ключа.

Защищён от прослушивания.

Не защищён от замены сообщений.

Основан на сложности задачи дискретного логарифмирования.

Протокол Диффи-Хеллмана



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

Протокол Диффи-Хеллмана

Ассиметричный протокол генерации симметричного ключа.

Защищён от прослушивания.

Не защищён от замены сообщений.

Основан на сложности задачи дискретного логарифмирования.

Существует версия алгоритма, основанная на эллиптических кривых.

Отношения между шифрованием и сжатием

Сначала сжимать, а потом шифровать, или наоборот?

Отношения между шифрованием и сжатием

Сначала сжимать, а потом шифровать, или наоборот?

Сначала сжимать, а потом шифровать!

Шифрованные файлы плохо сжимаются, потому что их энтропия высокая.

Имитовставка

Имитовставка

Имитовставка (MAC, англ. message authentication code — код аутентификации сообщения) — средство обеспечения имитозащиты в протоколах аутентификации сообщений с доверяющими друг другу участниками — специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности и аутентификации источника данных.

Имитовставка обычно применяется для обеспечения целостности и защиты от подделки передаваемой информации.

HMAC

HMAC (сокращение от англ. hash-based message authentication code, код аутентификации (проверки подлинности) сообщений, использующий хеш-функции) — один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними лицами.

Механизм HMAC использует MAC. Два клиента, использующие HMAC, как правило, разделяют общий секретный ключ. HMAC — надстройка над MAC; механизм обмена данными с использованием секретного ключа (как в MAC) и хеш-функций. В зависимости от используемой хеш-функции выделяют HMAC-MD5, HMAC-SHA1, HMAC-RIPEMD128, HMAC-RIPEMD160 и т. п.

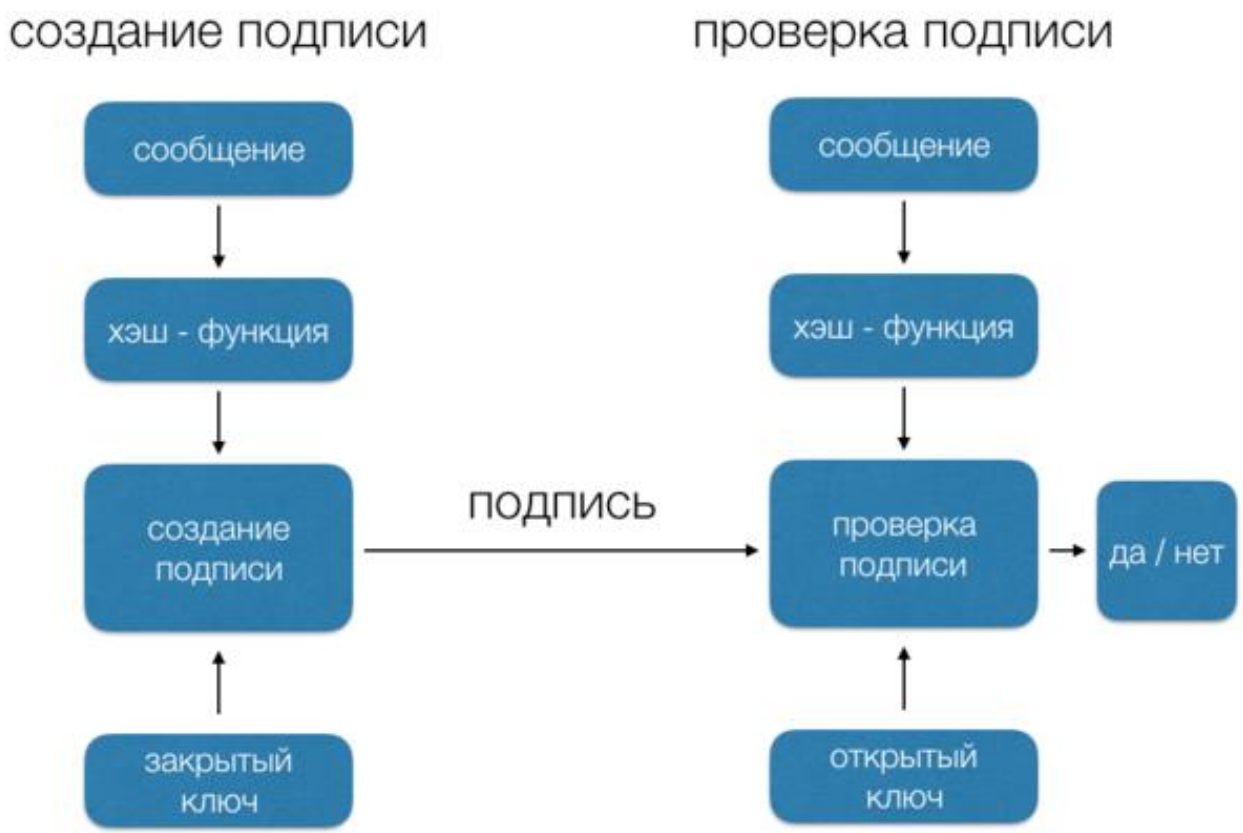
Одной формулой

$$\mathrm{HMAC}_K(text) = \mathrm{H}\left((K \oplus opad) \| \mathrm{H}\left((K \oplus ipad) \| text \right) \right)$$

«⊕» — операция «побитовое исключающее ИЛИ» или «xor»;
«||» — операция «склейка строк» (последовательностей байт);
opad, ipad — магические константы.

Электронная подпись

Механизм работы подписи



Ускорение работы подписи

Как мы видим из предыдущего слайда, подпись рассчитывается на основе хеша.

Это увеличивает скорость работы, но заставляет полагаться на стойкость хеш-функции.

DSA

DSA (англ. Digital Signature Algorithm — алгоритм цифровой подписи) — криптографический алгоритм с использованием открытого ключа для создания электронной подписи, но не для шифрования (в отличие от RSA и схемы Эль-Гамаля).

Алгоритм основан на вычислительной сложности взятия логарифмов в конечных полях.

ECDSA

ECDSA (Elliptic Curve Digital Signature Algorithm) — алгоритм с открытым ключом для создания цифровой подписи, аналогичный по своему строению DSA, но определённый, в отличие от него, не над кольцом целых чисел, а в группе точек эллиптической кривой.

Д. Брауном (Daniel R. L. Brown) было доказано, что алгоритм ECDSA не является более безопасным, чем DSA.

Самым важным преимуществом ECDSA является возможность его работы на значительно меньших полях. Для сравнения, при уровне безопасности в 80 бит (то есть атакующему необходимо примерно 2^{80} версий подписи для нахождения секретного ключа), размер открытого ключа DSA равен, по крайней мере, 1024 бит, тогда как открытого ключа ECDSA — 160 бит. С другой стороны размер подписи одинаков и для DSA, и для ECDSA: 4t бит, где t — уровень безопасности, измеренный в битах, то есть — примерно 320 бит для уровня безопасности в 80 бит.

RSA

Помимо шифрования, криптосистема RSA может использоваться для генерации цифровой подписи.

ГОСТ Р 34.10-2012

ГОСТ Р 34.10-2012 (полное название: «ГОСТ Р 34.10-2012. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи») — российский стандарт, описывающий алгоритмы формирования и проверки электронной цифровой подписи. Принят и введён в действие Приказом Федерального агентства по техническому регулированию и метрологии от 7 августа 2012 года № 215-ст вместо ГОСТ Р 34.10-2001. До ГОСТ Р 34.10-2001 действовал стандарт ГОСТ Р 34.10-94.

Основное отличие — в старом стандарте часть операций проводится над полем Z_p, а в новом — над группой точек эллиптической кривой

И кому нужен ГОСТ?

Случайные числа

Проблема

Генерация некачественных случайных чисел может привести к большой угрозе для криптографических алгоритмов.

Так, плохо выбранное большое простое число или сессионный ключ может привести к их предсказанию злоумышленником.

В ОС Linux

/dev/random — генератор случайных чисел;

/dev/urandom — генератор псевдослучайных чисел.

При чтении данных из устройства /dev/random выводятся только случайные байты, полностью состоящие из битов шума «хаотичного» пула ОС. Если «хаотичный» пул опустел, /dev/random ничего не выдаст, пока необходимое количество битов в пуле не будет создано, читающая /dev/random программа будет ждать появления очередного случайного байта.

В ядре Linux «хаотичный» пул получает энтропию из нескольких источников, в том числе из аппаратного генератора случайных чисел современных процессоров.

Устройство /dev/random может быть необходимо пользователям, которые требуют очень высокого коэффициента случайности, например, при создании ключа шифрования, предполагающего длительное использование.

Чтение данных устройства /dev/urandom возвратит столько байтов, сколько было запрошено. В результате, если в пуле было недостаточно битов, теоретически возможно найти уязвимость алгоритма, использующего это устройство/

И поэтому использовать надо…

… /dev/urandom подробности тут:

https://habr.com/ru/company/mailru/blog/273147/

А в Windows – CryptGenRandom/RtlGenRandom

Cryptographically Secure Randomness in C/C++

The easiest and safest solution here is to add libsodium as a dependency to your project and just use `randombytes_buf()`.

If this isn't considered an acceptable solution, take a close look at **how libsodium implements these functions**. The PHP team adopted a similar approach in **the internal `random_bytes` implementation**.

```c
#include "sodium.h"
int foo() {
    char myString[32];
    uint32_t myInt;

    randombytes_buf(myString, 32);
    /* myString will be an array of 32 random bytes, not null-terminated */
    myInt = randombytes_uniform(10);
    /* myInt will be a random number between 0 and 9 */
}
```

"Just use libsodium if you can," also applies for almost every other language below.

https://paragonie.com/blog/2016/05/how-generate-secure-random-numbers-in-various-programming-languages

НЕ ПРИДУМЫВАЙТЕ СВОИ КРИПТОАЛГОРИТМЫ

Почему?

Для того, чтобы создать действительно стойкий к атакам алгоритм, необходимо:

1. Огромный опыт в криптографии

2. Проверка результата другими людьми с огромным опытом в криптографии

3. Проверка временем

НЕ ПИШИТЕ СВОИ РЕАЛИЗАЦИИ КРИПТОАЛГОРИТМОВ

Почему?

Вы допустите ошибки в реализации. Неизбежно.

Что использовать?

1. Проверенные библиотеки, которые используют все
2. Штатные средства операционных систем

Не знаете, что использовать – используйте OpenSSL.

Рекомендуемая криптография для TLS

Order	Key Exchange Algorithm	Authentication Algorithm	Bulk Encryption Algorithm	Mac Algorithm
#1	Elliptic Curve Diffie–Hellman (ECDH)	Elliptic Curve Digital Signature Algorithm (ECDSA)	AES 256 in Galois Counter Mode (AES256-GCM)	SHA384
#2	Elliptic Curve Diffie–Hellman (ECDH)	RSA	AES 256 in Galois Counter Mode (AES256-GCM)	SHA384
#3	Elliptic curve Diffie–Hellman (ECDH)	Elliptic Curve Digital Signature Algorithm (ECDSA)	ChaCha20 (CHACHA20)	POLY1305
#4	Elliptic curve Diffie–Hellman (ECDH)	RSA	ChaCha20 (CHACHA20)	POLY1305
#5	Elliptic Curve Diffie–Hellman (ECDH)	Elliptic Curve Digital Signature Algorithm (ECDSA)	AES 128 in Galois Counter Mode (AES128-GCM)	SHA256