Хеш-функции и аутентификация сообщений

- Требования к криптографическим хеш-функциям
- Хеш-функции
  - Структура Меркля–Дамгарда
  - Хеш-функция MD5
  - Хеш-функция SHA-1
  - Хеш-функция SHA-2
  - Хеш-функция SHA-3
  - Хеш-функции ГОСТ 3411-94, 2012, 2018
- Коды аутентификации сообщений – MAC
  - Стандарт HMAC
  - Стандарт Poly1305
  - Стандарт AEAD

Требования к криптографическим хеш-функциям

- Хеш-функцией называется односторонняя функция, предназначенная для получения дайджеста или «отпечатков пальцев» файла, сообщения или некоторого блока данных.
  - хеш-код создается функцией **H**:
  - **h = H (M)**
  - где **M** является сообщением произвольной длины и **h** является хеш-кодом фиксированной длины.
  - Рассмотрим требования, которым должна соответствовать хеш-функция для того, чтобы она могла использоваться в качестве аутентификатора сообщения.

Требования к криптографическим хеш-функциям

Чтобы хеш-функция **H** могла использоваться в качестве аутентификатора сообщения, она должна обладать следующими свойствами:

1. хеш-функция **H** должна применяться к блоку данных любой длины.
2. хеш-функция **H** должна создавать выход фиксированной длины.
3. **H(M)** относительно легко (за полиномиальное время) вычисляется для любого значения **M**.
4. Для любого данного значения хеш-кода **h** вычислительно невозможно найти **M** такое, что **H(M)=h**.
5. Для любого данного **M** вычислительно невозможно найти **M′≠ M** такое, что **H(M)=H(M′).**
6. Вычислительно невозможно найти произвольную пару **(M,M′)** такую, что **H(M)=H(M′).**

Требования к криптографическим хеш-функциям

- Первые три свойства требуют, чтобы хеш-функция создавала хеш-код для любого сообщения.

Атака нахождения первого прообраза

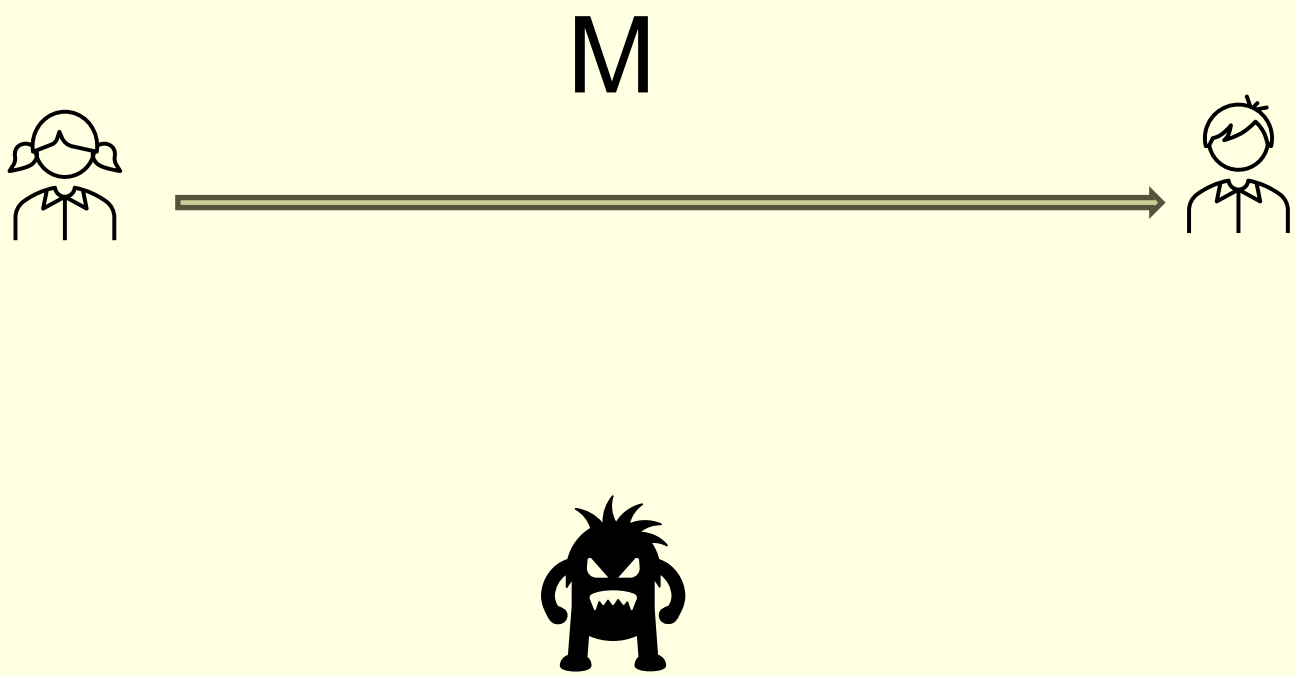
- Четвертое свойство означает, что хеш-функция должна обладать свойством односторонности: легко создать хеш-код по данному сообщению, но невозможно восстановить сообщение по хеш-коду. Это свойство важно, если для аутентификации с помощью хеш-функции используется секретное значение. Само секретное значение может не посылаться, тем не менее, если хеш-функция не является односторонней, противник может легко раскрыть секретное значение следующим образом. Перехватив передаваемое сообщение, атакующий получает сообщение **M** и хеш-код **h=H(S||M).** Если атакующий может инвертировать хеш-функцию, то он получает $\mathbf{S||M=H^{-1}(h)}$. Так как атакующий теперь знает и **M, и S||M**, получить **S** совсем просто.

Требования к криптографическим хеш-функциям

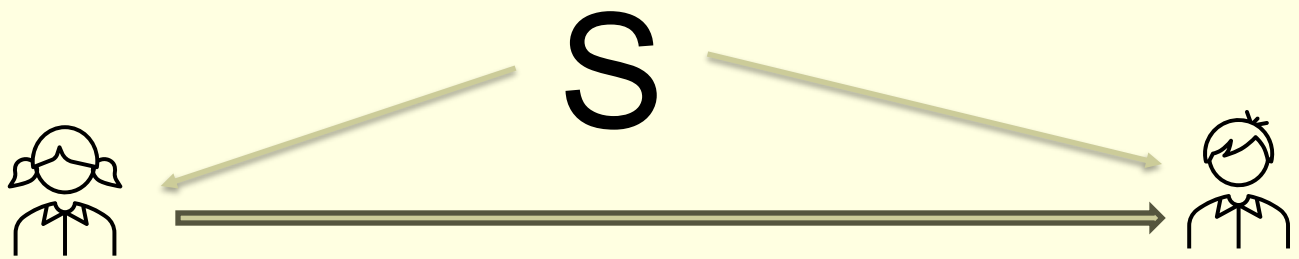
Атака нахождения второго прообраза

- Пятое свойство гарантирует, что невозможно найти другое сообщение, чье значение хеш-функции совпадало бы со значением хеш-функции данного сообщения. Это предотвращает подделку сообщения, когда в качестве аутентификатора используется защищенный от изменения хеш-код. Предположим, что противник может прочитать сообщение и, следовательно, вычислить его хеш-код. Но так как противник не знает секретного ключа, он не имеет возможности изменить сообщение так, чтобы получатель этого не обнаружил. Если данное свойство не выполняется, атакующий может выполнить следующую последовательность действий: перехватить сообщение и его зашифрованный хеш-код, вычислить хеш-код сообщения, создать альтернативное сообщение с тем же самым хеш-кодом, заменить исходное сообщение на поддельное. Поскольку хеш-коды этих сообщений совпадают, получатель не обнаружит подмены.

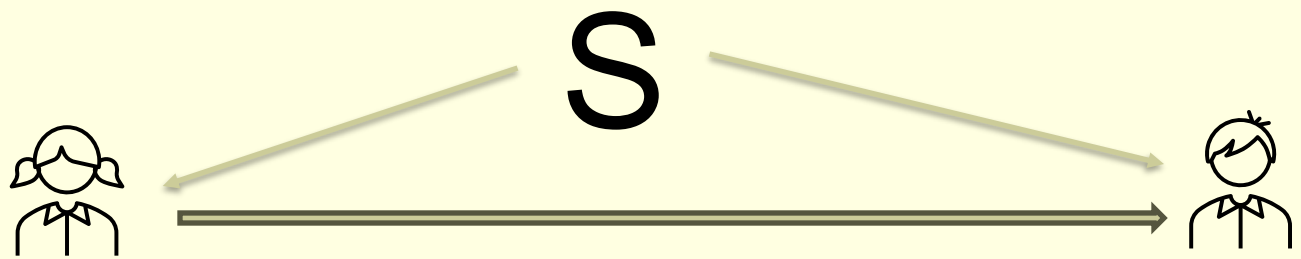
Использование криптографической хеш-функции для аутентификации и обеспечения целостности



Использование криптографической хеш-функции для аутентификации и обеспечения целостности

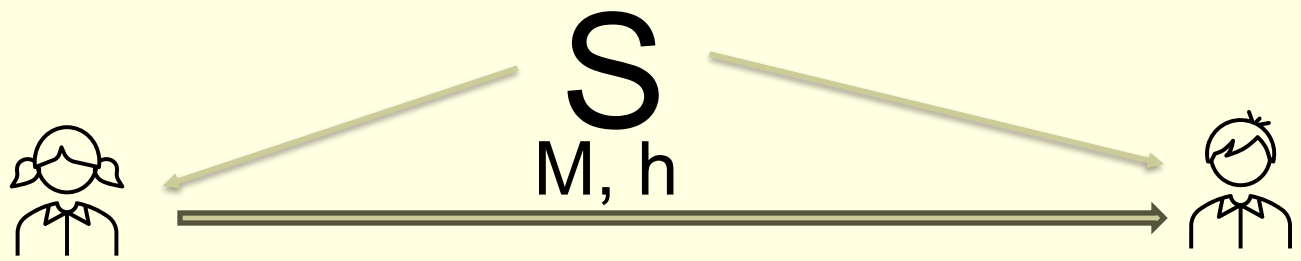


Использование криптографической хеш-функции для аутентификации и обеспечения целостности



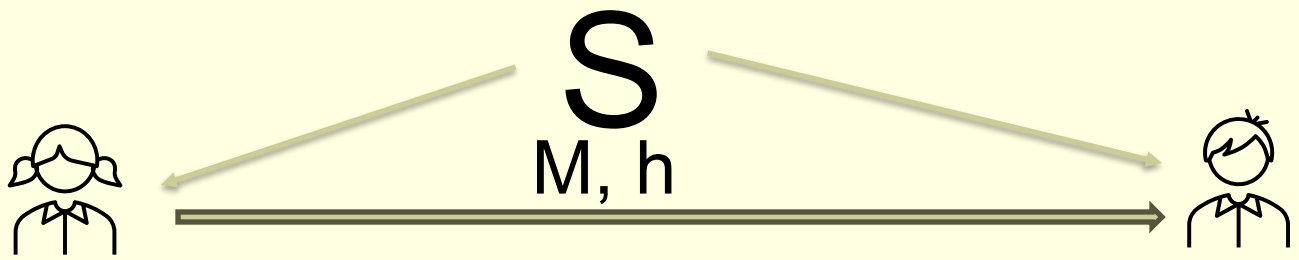
$h = H(M||S)$

Использование криптографической хеш-функции для аутентификации и обеспечения целостности



$h = H(M||S)$

Использование криптографической хеш-функции для аутентификации и обеспечения целостности



$h = H(M||S)$

$h' = H(M' ||S)$

Если $h = h'$:

сообщение не изменено и получено от того, кто знает S

Требования к криптографическим хеш-функциям

Атака день рождения

- Шестое свойство защищает против класса атак, известных как «**Атака день рождения**».

Требования к криптографическим хеш-функциям

Атака нахождения второго прообраза

- Первая задача. Каким должно быть число **k**, чтобы для данного значения **X** и значений $Y_1, \ldots, Y_k$, каждое из которых принимает значения от **1** до **n**, вероятность того, что хотя бы для одного Y_i выполнялось равенство **X=Y**

$$P\ (X = Y) \geq 0.5$$

- Для одного значения **Y** вероятность того, что **X=Y**, равна **1/n**.

$$P\ (X = Y) = 1/n$$

- Соответственно, вероятность того, что **X≠Y**, равна **1 – 1/n**.

$$P\ (X \neq Y) = 1 - 1/n$$

- Если создать **k** значений, то вероятность того, что ни для одного из них не будет совпадений, равна произведению вероятностей, соответствующих одному значению, т.е. $(1 - 1/n)^k$.

Требования к криптографическим хеш-функциям

- Следовательно, вероятность по крайней мере одного совпадения равна

$$P(X = Y_i) = 1 - (1 - 1/n)^k$$

По формуле бинома Ньютона

$$(1 - a)^k = 1 - ka + \frac{k(k-1)}{2!} a^2 - \ldots \approx 1 - ka$$

$$1 - (1 - k/n) = k/n = 0.5$$

$$k = n/2$$

Таким образом, для хэш-кода длиной m бит достаточно выбрать 2^{m-1} сообщений, чтобы вероятность совпадения хэш-кодов была больше 0,5.

Требования к криптографическим хеш-функциям

Атака Дня рождения

- Теперь рассмотрим вторую задачу. Обозначим **P(n,k)** вероятность того, что в множестве из **k** элементов, каждый из которых может принимать **n** значений, есть хотя бы два с одинаковыми значениями. Чему должно быть равно **k**, чтобы **P(n,k)** была бы больше 0,5?
- Число различных способов выбора элементов таким образом, чтобы при этом не было дублей, равно

$$n\ (n-1)\ \ldots\ (n-k+1) = \frac{n!}{(n-k)!}$$

Всего возможных способов выбора элементов равно n^k

Требования к криптографическим хеш-функциям

Вероятность того, что дублей нет, равна

$$\frac{n!}{(n-k)!\ n^k}$$

Вероятность того, что есть дубли, соответственно равна

$$1 - \frac{n!}{(n-k)!\ n^k}$$

$$P(n, k) = 1 - n! / ((n-k)! \times n^k) = 1 - (n \times (n-1) \times \ldots \times (n-k-1)) / n^k =$$

$$1 - [\ (n-1)/n \times (n-2)/n \times \ldots \times (n-k+1)/n] =$$

$$1 - [(\ 1-\ 1/n) \times (1 - 2/n) \times \ldots \times (1 - (k-1)/n)]$$

Требования к криптографическим хеш-функциям

Известно, что

$$1 - x \leq e^{-x}$$

$$P(n, k) > 1 - [e^{-1/n} \times e^{-2/n} \times \ldots \times e^{-k/n}]$$

$$P(n, k) > 1 - e^{-k(k-1)/n}$$

$$\frac{1}{2} = 1 - e^{-k(k-1)/n}$$

$$2 = e^{k(k-1)/n}$$

$$\ln 2 = k(k-1) / 2n$$

$$k(k-1) \approx k^2$$

$$k = \sqrt{(2n \times \ln 2)} = 1{,}17\sqrt{n} \approx \sqrt{n}$$

Требования к криптографическим хеш-функциям

- Если хэш-код имеет длину $\mathbf{m}$ бит, т.е. принимает $\mathbf{2^m}$ значений, то

$$\mathbf{k = \sqrt{2^m} = 2^{m/2}}$$

- Подобный результат называется «парадоксом дня рождения», потому что в соответствии с приведенными выше рассуждениями для того, чтобы вероятность совпадения дней рождения у двух человек была больше 0,5, в группе должно быть всего 23 человека. Этот результат кажется удивительным, возможно, потому, что для каждого отдельного человека вероятность того, что с его днем рождения совпадет день рождения у кого-то другого в группе, достаточно мала.

Требования к криптографическим хеш-функциям

- Вернемся к рассмотрению свойств хеш-функций. Предположим, что используется 64-битный хеш-код. Можно считать, что это вполне достаточная и, следовательно, безопасная длина для хеш-кода. Например, если защищенный хеш-код **h** передается с соответствующим незашифрованным сообщением **M**, то противнику необходимо будет найти **M'** такое, что

$$H(M') = H(M)$$

- для того, чтобы подменить сообщение и обмануть получателя. В среднем противник должен перебрать 2^{63} сообщений для того, чтобы найти другое сообщение, у которого хеш-код равен перехваченному сообщению.

Требования к криптографическим хеш-функциям

- Тем не менее, возможны различного рода атаки, основанные на «парадоксе дня рождения». Возможна следующая стратегия:

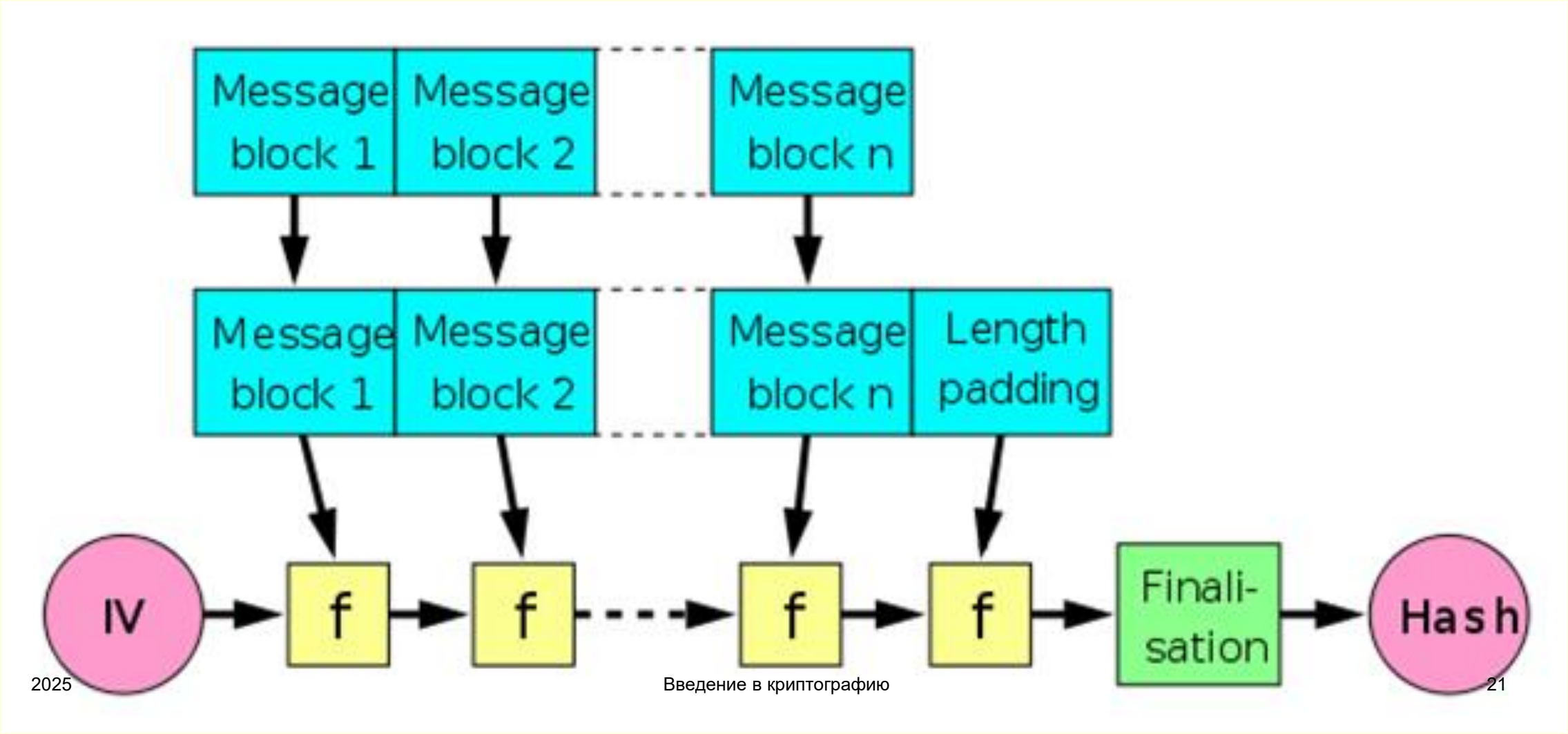
1. Противник создает $\mathbf{2^{m/2}}$ вариантов сообщения, каждое из которых имеет некоторый определенный смысл. Противник подготавливает такое же количество сообщений, каждое из которых является поддельным и предназначено для замены настоящего сообщения.
2. Два набора сообщений сравниваются в поисках пары сообщений, имеющих одинаковый хеш-код. Вероятность успеха в соответствии с «парадоксом дня рождения» больше, чем 0,5. Если соответствующая пара не найдена, то создаются дополнительные исходные и поддельные сообщения до тех пор, пока не будет найдена пара.

Требования к криптографическим хеш-функциям

3. Атакующий предлагает отправителю исходный вариант сообщения для подписи. Эта подпись может быть затем присоединена к поддельному варианту для передачи получателю. Так как оба варианта имеют один и тот же хеш-код, подпись будет соответствовать обоим сообщениям. Противник подписал поддельное сообщение, не зная при этом ключа, защищающего хеш-код.

- Таким образом, если используется 64-битный хеш-код, то для подбора двух сообщений с одинаковым хеш-кодом в среднем необходимо перебрать 2^{32} сообщений.
- Вследствие этого длина хеш-кода должна быть достаточно большой. Длина, равная 64 битам, в настоящее время не считается безопасной. Предпочтительнее, чтобы длина составляла не менее 100 битов.

Структура Меркля–Дамгарда



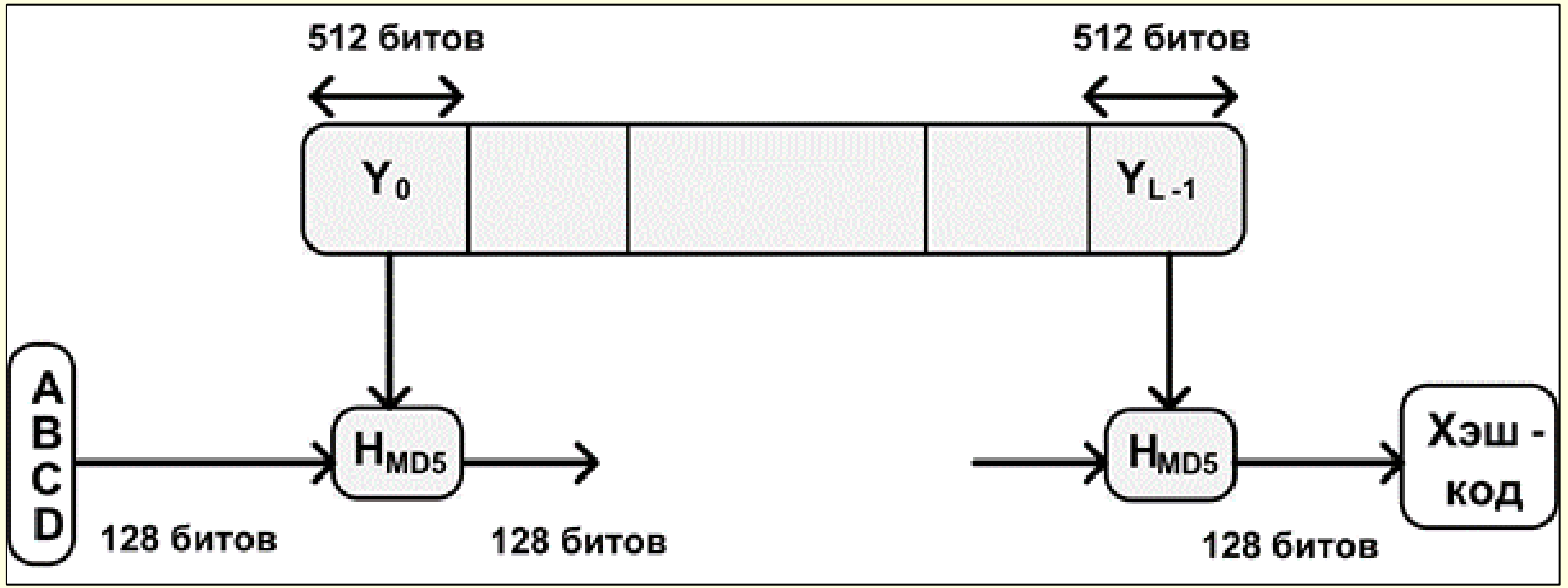
Структура Меркля–Дамгарда

- **Структура Меркля — Дамгарда** (*MD*, *Merkle—Damgård*) — метод построения криптографических хеш-функций, предусматривающий разбиение входных сообщений произвольной длины на блоки фиксированной длины и работающий с ними по очереди с помощью функции сжатия, каждый раз принимая входной блок с выходным от предыдущего прохода.
- Меркл и Дамгард показали, что если функция сжатия устойчива к коллизиям, то и хеш-функция будет также устойчива. Для повышения устойчивости к коллизиям сообщение дополняется блоком, который кодирует длину первоначального сообщения (упрочнение Меркля — Дамгарда).

Структура Меркля–Дамгарда

- Односторонняя функция сжатия **f** преобразует два входных блока фиксированной длины в выходной блок того же размера, что и входные; алгоритм начинает с вектора инициализации IV, функция **f** выполняется последовательно над результатом каждого предыдущего прохода.
- Структура предусматривает вектор инициализации — фиксированное значение, которое зависит от реализации алгоритма, и которое применяется к первому проходу — применению функции сжатия **f** к нему и первому блоку сообщения. Результат каждого прохода передаётся на следующий вход **f** и очередному блоку сообщения; последний блок дополняется нулями, если необходимо, также, добавляется блок с информацией о длине целого сообщения. Последний результат иногда пропускают через функцию финализации (*finalisation function*), которая может использоваться также для уменьшения размера выходного хеша сжатием результата последнего применения **f** в хеш более маленького размера, или чтобы гарантировать лучшее смешивание битов и усилить влияние небольшого изменения входного сообщения на хеш (обеспечить лавинный эффект). Функция финализации часто строится с использованием функции сжатия.
- Основные алгоритмы, реализующие структуру Меркля-Дамгарда — MD5, SHA-1, семейство SHA-2.

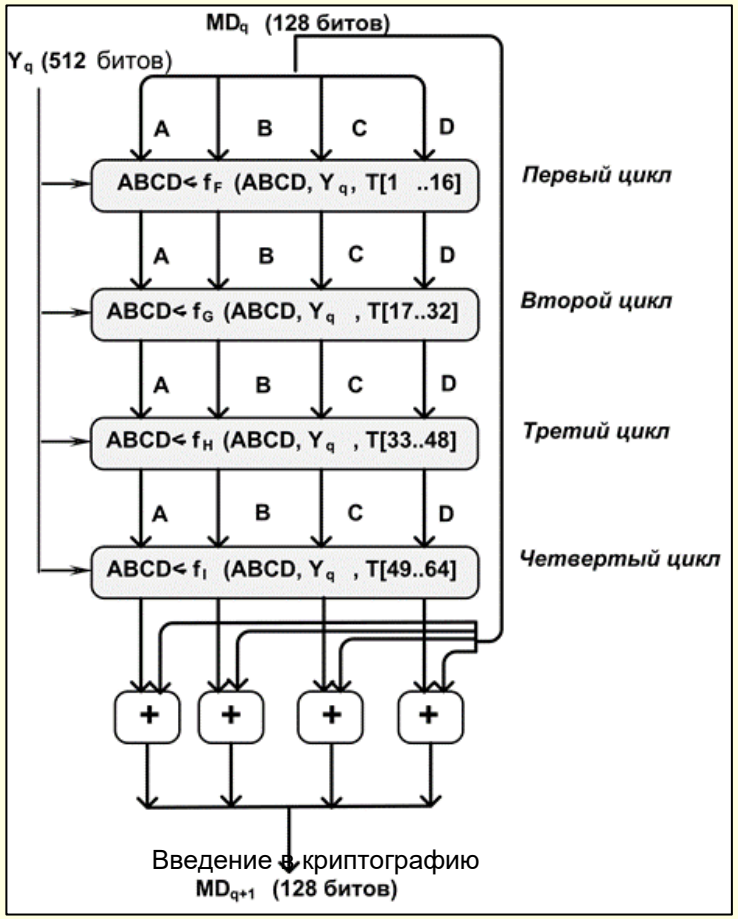
Хеш-функция MD5




```
A = 01234567
B = 89ABCDEF
C = FEDCBA98
D = 76543210
```

Хеш-функция MD5

Обработка очередного 512-битного блока



Хеш-функция MD5

Логика выполнения отдельного шага

$A \leftarrow B + CLS_s\ (A + f\ (B, C, D) + X\ [k] + T\ [i])$

где

A, B, C, D - четыре слова буфера; после выполнения каждого отдельного шага происходит циклический сдвиг влево на одно слово.

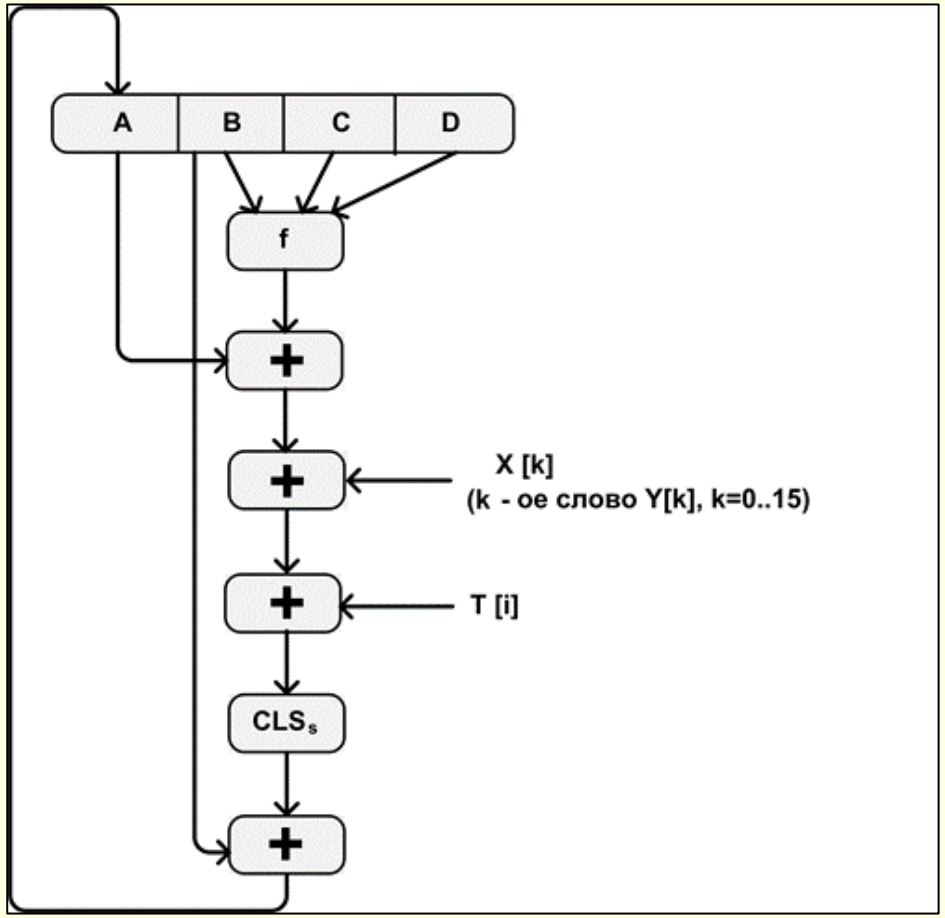
f - одна из элементарных функций f_F, f_G, f_H, f_I.

CLS_s - циклический сдвиг влево на **s** битов 32-битного аргумента.

X[k] - M[q*16 + k] - **k**-ое 32-битное слово в **q**-ом 512 блоке сообщения.

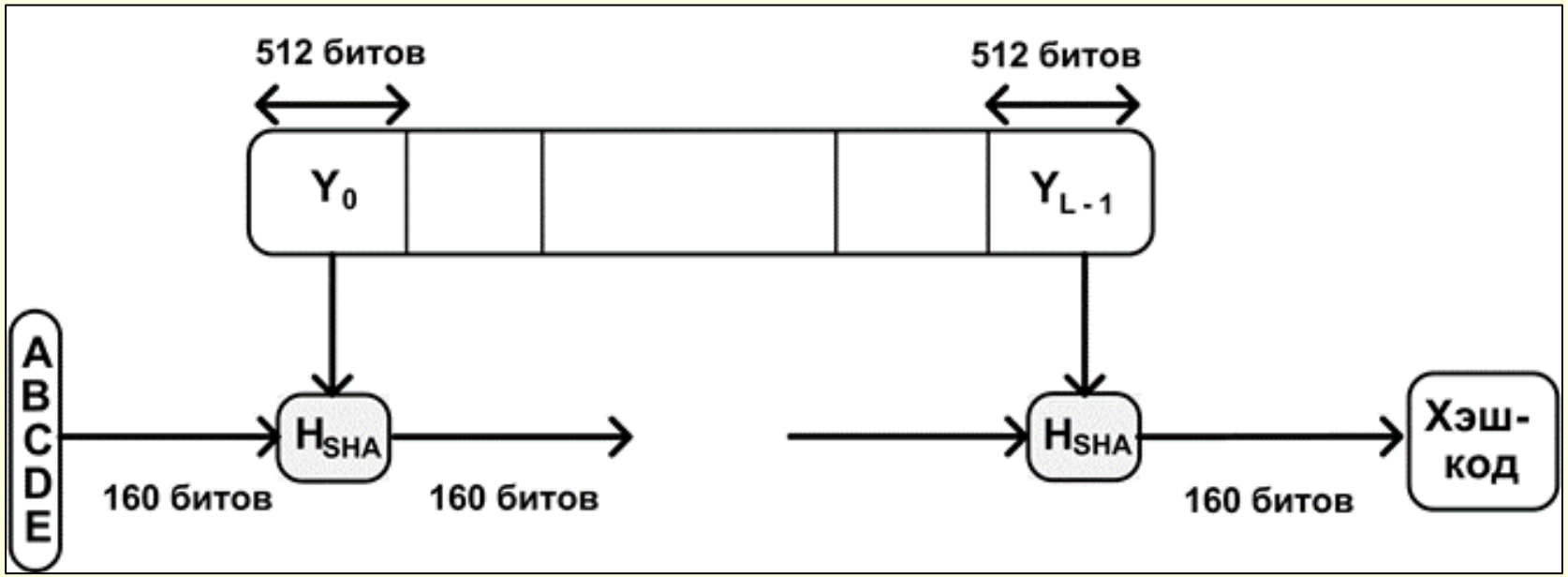
T[i] - **i**-ое 32-битное слово в таблице **T**.

+ - сложение по модулю 2^{32}.



Хеш-функция SHA-1

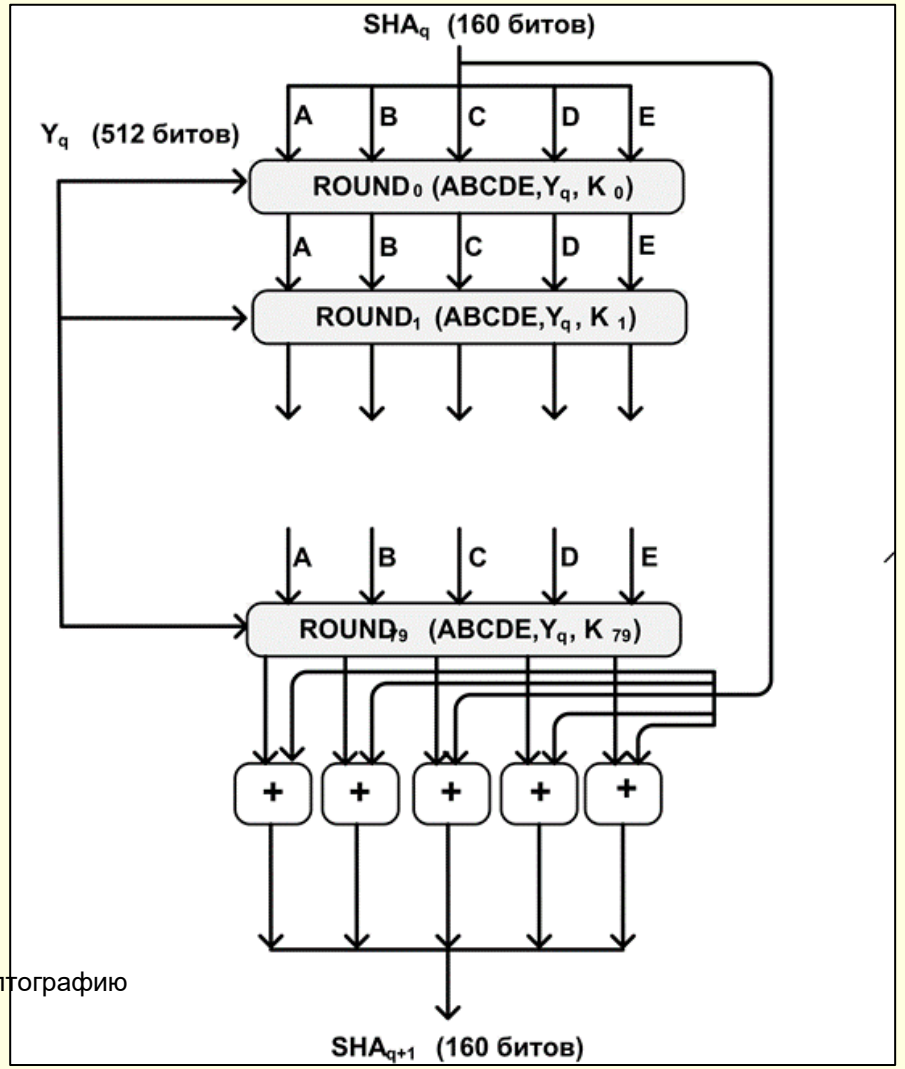
- Безопасный хеш-алгоритм (Secure Hash Algorithm) был разработан NIST и опубликован в качестве федерального информационного стандарта (FIPS PUB 180) в 1993 году. У алгоритмов MD5 и SHA-1 много общего.




```
A = 67452301
B = EFCDAB89
C = 98BADCFE
D = 10325476
E = C3D2E1F0
```

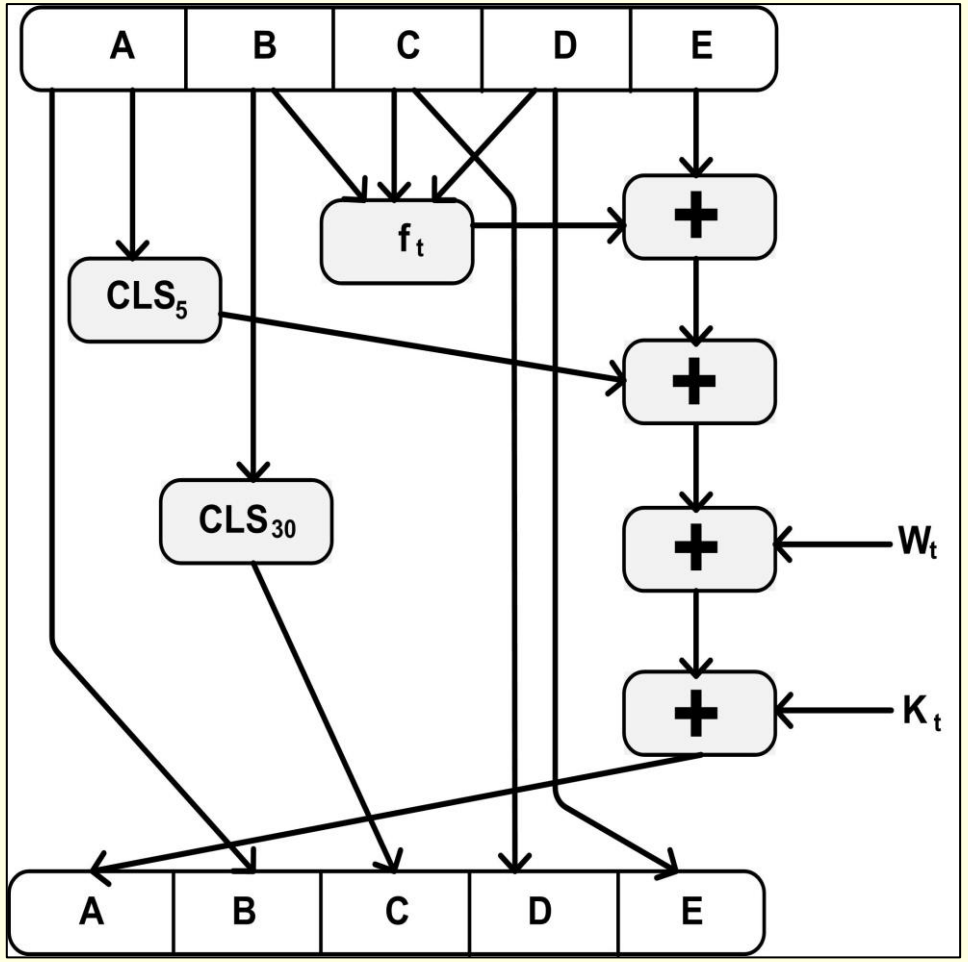
Хеш-функция SHA-1

Обработка очередного 512-битного блока



Хеш-функция SHA-1

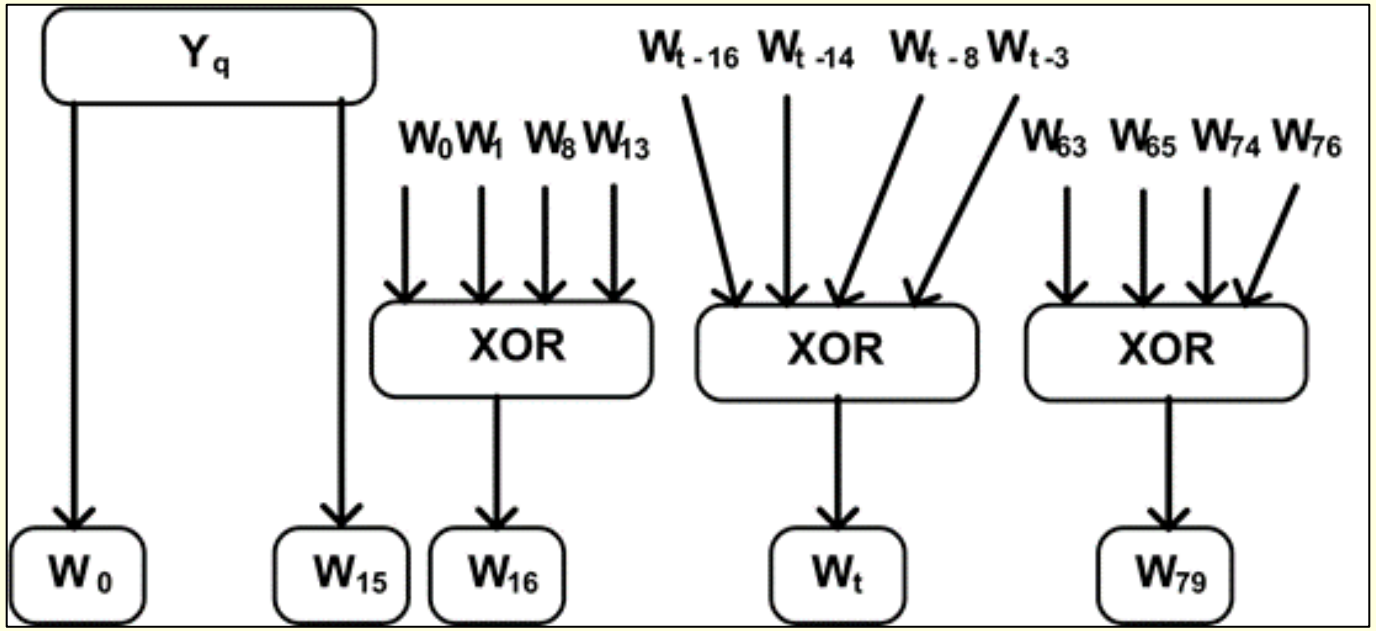
Логика выполнения отдельного цикла



Хеш-функция SHA-1

Получение входных значений каждого цикла из очередного блока

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$$



Хеш-функция SHA-1

Сравнение SHA-1 и MD5

	MD5	SHA-1
Длина дайджеста	128 бит	160 бит
Размер блока обработки	512 бит	512 бит
Число итераций	64 (4 цикла по 16 итераций в каждом)	80
Число элементарных логических функций	4	3
Число дополнительных констант	64	4

Хеш-функции SHA-2

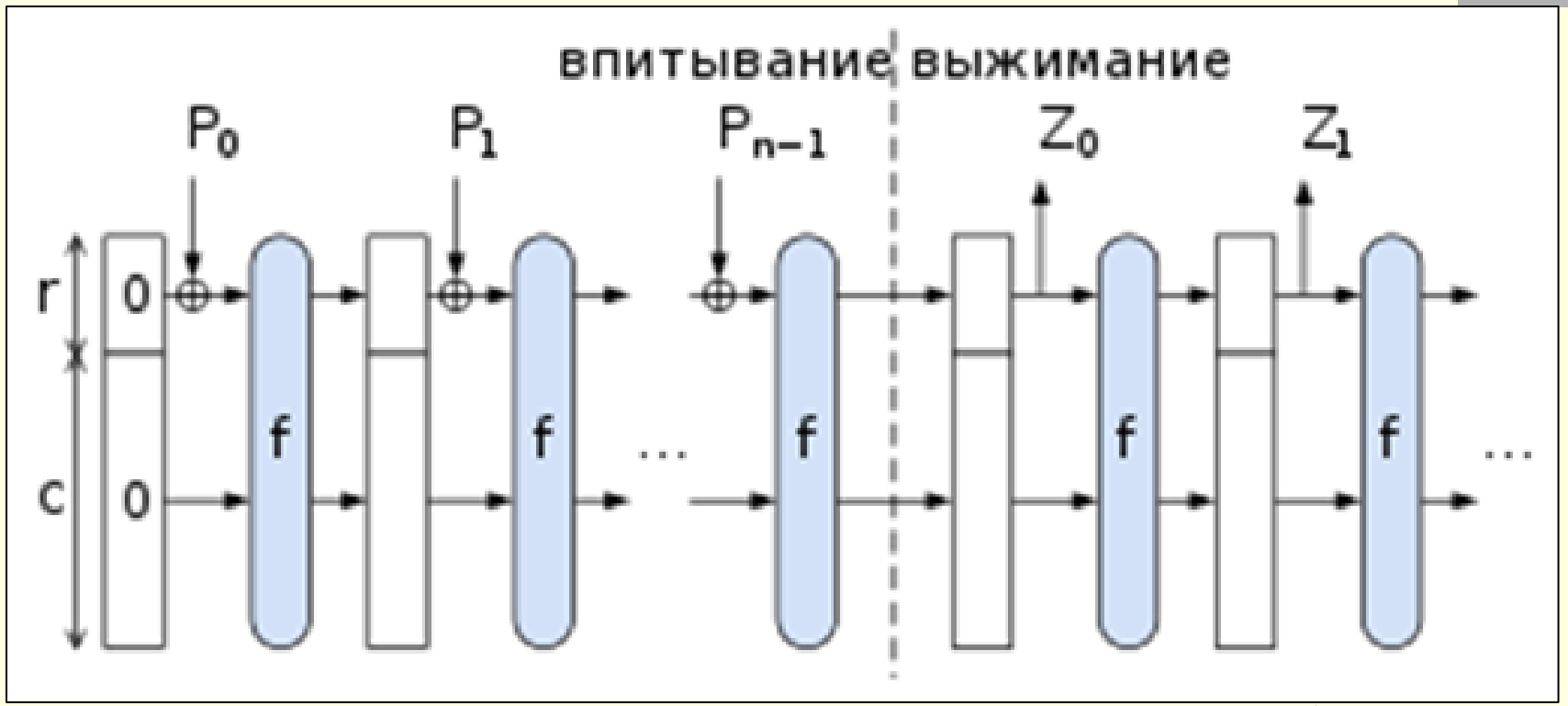
- В 2001 году NIST принял в качестве стандарта три хеш-функции с существенно большей длиной хеш-кода. Часто эти хеш-функции называют SHA-2 или SHA-256, SHA-384 и SHA-512 (соответственно, в названии указывается длина создаваемого ими хеш-кода). Эти алгоритмы отличаются не только длиной создаваемого хеш-кода, но и длиной обрабатываемого блока, длиной слова и используемыми внутренними функциями.

Алгоритм	Длина сообщения (в битах)	Длина блока (в битах)	Длина слова (в битах)	Длина дайджеста сообщения (в битах)	Безопасность (в битах)
SHA-1	$<2^{64}$	512	32	160	80
SHA-256	$<2^{64}$	512	32	256	128
SHA-384	$<2^{128}$	1024	64	384	192
SHA-512	$<2^{128}$	1024	64	512	256

Хеш-функции SHA-3

SHA-3 (*Keccak* — произносится как «кечак») — алгоритм хеширования переменной разрядности, разработанный группой авторов во главе с Йоаном Дайменом, соавтором Rijndael. В 2012 году Keccak стал победителем конкурса криптографических алгоритмов, проводимым NIST. В 2015 году алгоритм утверждён и опубликован в качестве стандарта FIPS 202.

Хеш-функции SHA-3



SHA-3 использует конструкцию криптографической губки

P_i — входные блоки, Z_j — выход алгоритма. Размер c («capacity») не используемого для получения выходного набора битов должен быть значительным для предотвращения атак.

Хеш-функции SHA-3

Хеш-функции семейства *SHA-3* построены на основе конструкции криптографической губки, в которой данные сначала «впитываются» в губку, при котором исходное сообщение **M** подвергается многораундовым перестановкам **f**, затем результат **Z** «отжимается» из губки. На этапе «впитывания» блоки сообщения суммируются по модулю 2 с подмножеством состояния, которое затем преобразуется с помощью функции перестановки **f**. На этапе «отжимания» выходные блоки считываются из одного и того же подмножества состояния, изменённого функцией перестановок **f**. Размер части состояния, который записывается и считывается, называется «скоростью» (rate) и обозначается **r**, а размер части, которая не тронута вводом/выводом, называется «емкостью» (capacity) и обозначается **c**.

Хеш-функции SHA-3

Алгоритм получения значения хеш-функции можно разделить на несколько этапов:

- Исходное сообщение **M** дополняется до строки **P** длины, кратной **r**, с помощью функции дополнения (pad-функции)
- Строка **P** делится на **n** блоков длины **r**: $\mathbf{P_0}, \mathbf{P_1}, \ldots, \mathbf{P_{n-1}}$
- **«Впитывание»:** каждый блок $\mathbf{P_i}$ дополняется нулями до строки длины **b** бит и суммируется по модулю 2 со строкой состояния **S**, где **S** — строка длины **b** бит

$\mathbf{b = r + c}$

- Перед началом работы функции все элементы **S** равны нулю. Для каждого следующего блока состояние — строка, полученная применением функции перестановок **f** к результату предыдущего шага
- **«Отжимание»:** пока длина **Z** меньше **d, d** — количество бит в результате хеш-функции), к **Z** добавляется **r** первых бит состояния **S**, после каждого прибавления к **S** применяется функция перестановок **f**. Затем **Z** обрезается до длины **d** бит
- Строка **Z** длины **d** бит возвращается в качестве результата

Хеш-функции SHA-3

- Благодаря тому, что состояние содержит **c** дополнительных бит, алгоритм устойчив к атаке удлинением сообщения, к которой восприимчивы алгоритмы SHA-1 и SHA-2.
- Состояние обозначается следующим образом:

  S = S[0] || S[1] || … || S[b-1]
- В SHA-3 состояние **S** является массивом 5 × 5 слов длиной **w** = 64 бита, всего 5 × 5 × 64 = 1600 бит. Также в Keccak могут использоваться длины w, равные меньшим степеням 2 (от w = 1 до w = 32).

Хеш-функции SHA-3

Добавление

- Для того, чтобы исходное сообщение ***M*** можно было разделить на блоки длины ***r***, необходимо добавление. В SHA-3 используется шаблон добавления **<10*1>**: к сообщению добавляется 1, после него 0 или больше нулевых битов (до *r-1*), в конце 1.
- *r-1* нулевых битов может быть добавлено, когда последний блок сообщения имеет длину *r-1* бит. Этот блок дополняется единицей, следующий блок будет состоять из *r-1* нулей и единицы.
- Два единичных бита добавляются и в том случае, если длина исходного сообщения *M* делится на *r*. В этом случае к сообщению добавляется блок, начинающийся и оканчивающийся единицами, между которыми *r-2* нулевых битов. Это необходимо для того, чтобы для сообщения, оканчивающегося последовательностью битов как в функции дополнения, и для сообщения без этих битов значения хеш-функции были различны.
- Первый единичный бит необходим для того, чтобы результаты хеш-функции от сообщений, отличающихся несколькими нулевыми битами в конце, были различны.

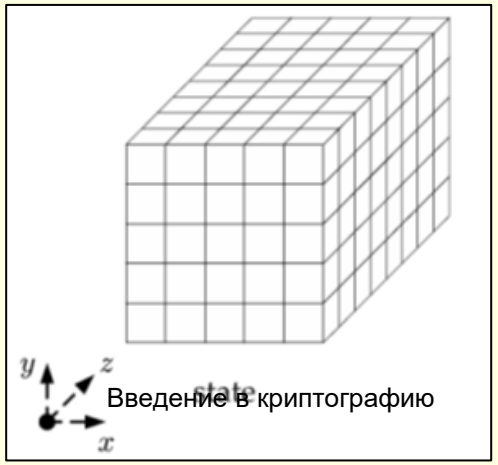
Хеш-функции SHA-3

Функция перестановок

- Функция перестановок, используемая в SHA-3, включает в себя **исключающее «ИЛИ» (XOR)**, **побитовое «И» (AND)** и **побитовое отрицание (NOT)**. Функция определена для строк длины, равной степени 2. $w = 2^l$. В основной реализации SHA-3 $w = 64$ ($l=6$).
- Состояние обозначается следующим образом:

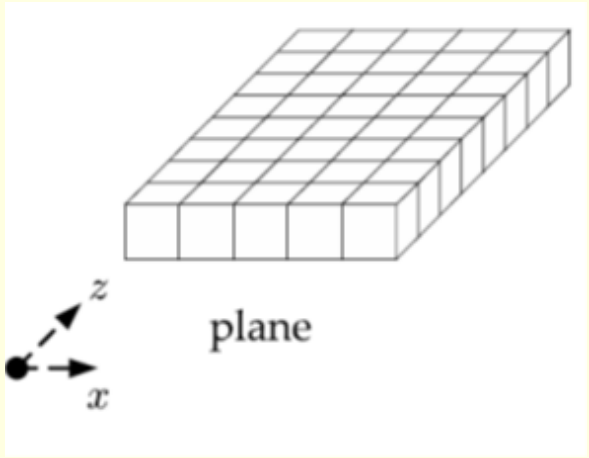
  S = S[0] || S[1] || … || S[b-1]

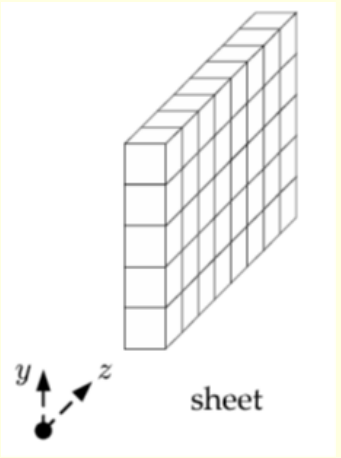
- Состояние **S** можно представить в виде трёхмерного массива **A** размером $5 \times 5 \times w$. Тогда элемент массива A[i][j][k] – это $(5i + j) \times w + k$ бит строки состояния **S**.

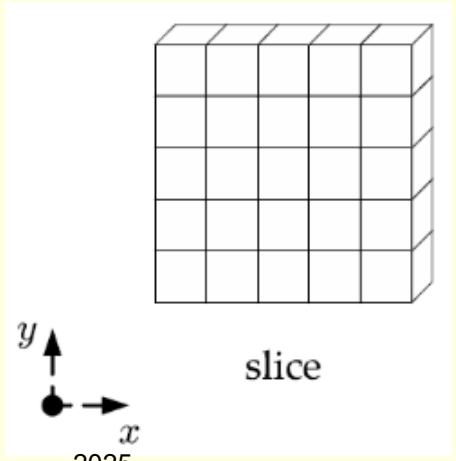


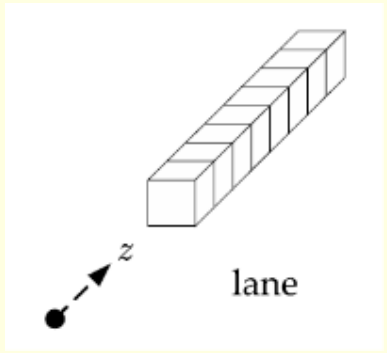
Состояние S

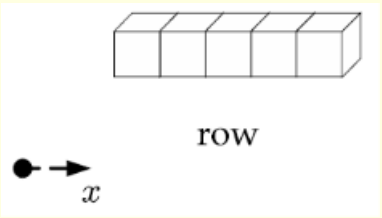
Хеш-функции SHA-3

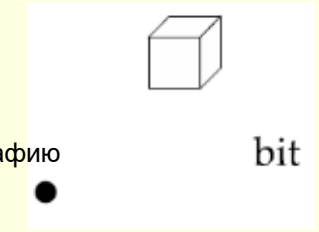


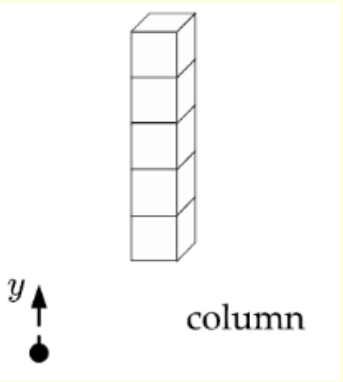












Хеш-функции SHA-3

- Функция содержит несколько шагов: θ, ρ, π, χ, ι, которые выполняются несколько раундов. На каждом шаге обозначим входной массив A, выходной массив A'.
- Для всех троек таких, что 0 <= x < 5, 0 <= y < 5, 0 <= z < w

$$\mathbf{A\,[x, y, z] = S\,[w(5y + x) + z]}$$

Хеш-функции SHA-3

Спецификация θ

Вход: массив состояния A

Выход: массив состояния A′

Шаги:

1. Для всех пар (x,z), таких, что 0 <= x < 5, 0 <= z < w,

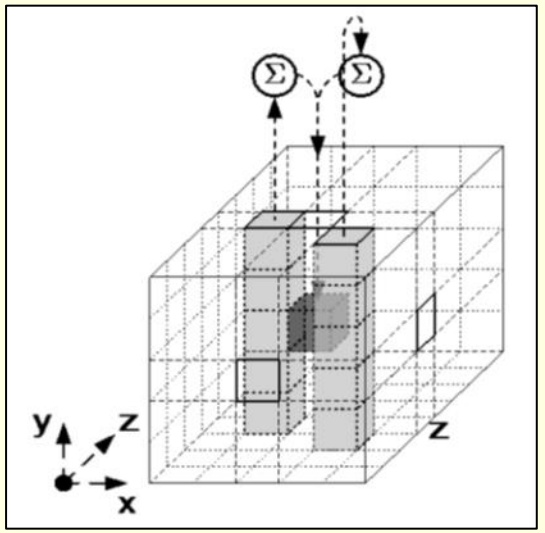
C [x,z] = A[x, 0, z] ⊕ A [x, 1 z] ⊕ A [x, 2, z] ⊕ A [x, 4, z]

2. Для всех пар (x,z), таких, что 0 <= x < 5, 0 <= z < w,

D [x,z] = C [(x-1) mod 5, z] ⊕ C [(x+1) mod 5, (z-1) mod w]

3. Для всех (x, y, z), таких, что 0 <= x < 5, 0 <= y < 5, 0 <= z < w,

A′ [x, y, z] = A [x, y, z] ⊕ D [x, z]

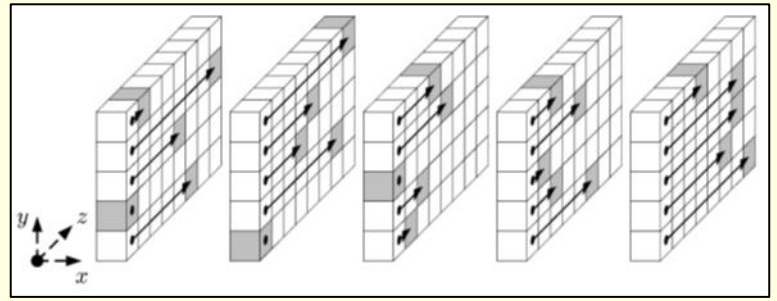


Применение θ к одному биту [8]

Результатом θ является XOR каждого бита в состоянии с четностью двух столбцов в массиве. На рисунке символ Σ означает четность, т.е. сумму XOR всех битов в столбце.

Хеш-функции SHA-3

Спецификация ρ



Вход: массив состояния A

Выход: массив состояния A′

Шаги:

a) Для всех z, таких, что 0 <= z < w, A′ [0, 0, z] = A [0, 0, z]
b) Пусть в начале (x, y) = (1, 0).
c) Для t от 0 до 23:
    a) Для всех z, таких, что 0 <= z < w, **A′ [x, y, z] = A [x, y, (z – (t+1)(t+2)/2) mod w]**
    b) **(x, y) = (y, (2x + 3y) mod 5)**
4. Вернуть A′

Хеш-функции SHA-3

- Результат **ρ состоит в ротации битов для каждой полосы (lane)** длиной, называемой *offset*, которая зависит от фиксированных координат x и y полосы. Это эквивалентно тому, что для каждого бита в полосе координата z изменяется добавлением offset по модулю размера полосы.
- На рисунке приведен результат применения ρ для случая w=8.
- Для полосы на рисунке черная точка обозначает бит, координата которого z равна 0, затененный куб указывает позицию, которую будет иметь бит после применения ρ. Другие биты в полосе сдвигаются на тоже самое значение, сдвиг является циклическим. Например, сдвиг для полосы A [1,0] равен 1, поэтому последний бит, чья координата z равна 7 для данного примера, сдвигается на переднюю позицию, чья координата z равна 0. Следовательно, сдвиги могут быть понижены по модулю размера полосы; например, полоса для A[3,2] в верхней части крайнего левого листа имеет смещение 153 mod 8 для данного примера, т.е. смещение равно 1 биту.

Хеш-функции SHA-3

Спецификация π

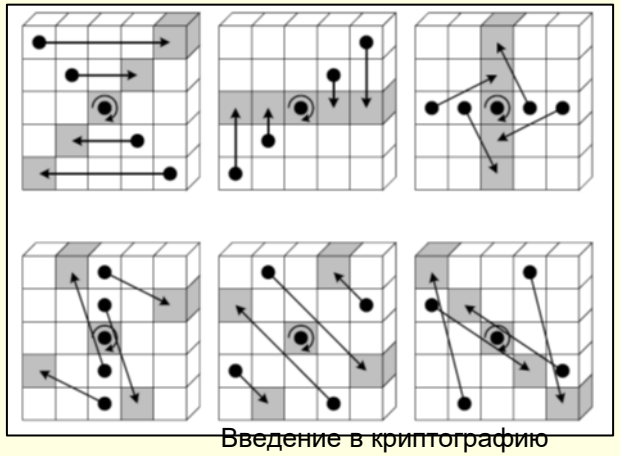
Шаги:

1. Для всех троек (x, y, z), таких, что 0 <= x < 5, 0 <= y < 5, 0 <= z < w

$$A'[x, y, z] = A[(x + 3y) \bmod 5, x, z]$$

2. Вернуть A′

Результат **π состоит в перестановке позиций полос**, как показано на рисунке для любого слоя.



Хеш-функции SHA-3

Спецификация χ

Шаги:

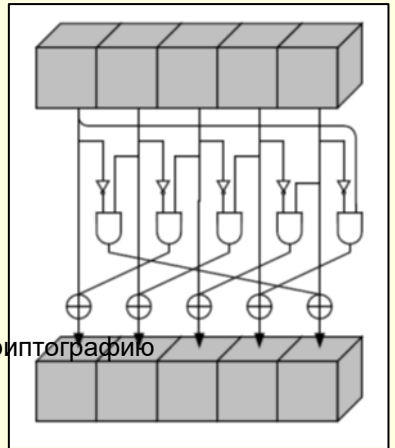
1. Для всех (x, y, z), таких, что 0 <= x < 5, 0 <= y <5, 0 <= z < w

$$A'[x, y, z] = A[x, y, z] \oplus ((A[(x+1) \bmod 5, y, z] \oplus 1) \cdot A[(x+2) \bmod 5, y, z])$$

2. Вернуть A′

Точка в правой части уравнения шага 1 обозначает умножение целых, которое в данном случае эквивалентно булевской операции «AND».

Результат **χ состоит в XOR каждого бита с нелинейной функцией двух других битов в данной строке**, как показано на рисунке.



Хеш-функции SHA-3

Спецификация ι

Отображение ι параметризуется индексом раунда i_r, чье значение определяется на шаге 2 алгоритма 7 вычисления Keccak-p [b, n_r], который будет описан далее. В самой спецификации ι в алгоритме 6 данный параметр определяет *l+1* биты значения полосы, называемые *константой раунда*, обозначаемой *RC*. Каждые из этих *l+1* бит создаются функцией, которая является основой линейного регистра сдвига с обратной связью. Данная функция, обозначаемая *rc*, специфицирована в алгоритме 5.

Алгоритм 5: *rc(t)*

Шаги:

Введем дополнительную функцию rc (t), где вход - целое число t, а на выходе бит.

Алгоритм rc (t)

1. Если t mod 255 = 0, то возвращается 1
2. Пусть R = 10000000
3. Для t от 1 до 255:
    1. R = 0 || R
    2. R[0] = R[0] ⊕ R[8]
    3. R[4] = R[4] ⊕ R[8]
    4. R[5] = R[5] ⊕ R[8]
    5. R[6] = R[6] ⊕ R[8]
    6. R = $Trunc_8$ [R]
4. Возвращается R[0]

Хеш-функции SHA-3

Алгоритм 6: ι (A, i_r)

i_r - номер раунда.

1. Для всех (x, y, z), таких, что 0 <= x < 5, 0 <= y < 5, 0 <= z < w A′ [x, y, z] = A [x, y, z]
2. Пусть RC - массив длины w, заполненный нулями.
3. Для j от 0 до l: RC $[2^j - 1]$ = rc $(j + 7i_r)$
4. Для всех z, таких, что 0 <= z < w, A′ [0, 0, z] = A′ [0, 0, z] ⊕ RC [z]

Результат ι состоит в модификации битов *Lane (0,0)* способом, который зависит от индекса раунда i_r. Остальные 24 полосы не затронуты ι.

Хеш-функции SHA-3

Алгоритм 7: Keccak-p [b, n_r]

1. Перевод строки S в массив A
2. Для i_r от $12 + 2l$ -n_r до $12 + 2l - 1$

$$\mathbf{A' = \iota\,(\chi\,(\pi\,(\rho\,(\theta\,(A)))), i_r)}$$

3. Перевод массива A′ в строку S длины b

Хеш-функции SHA-3

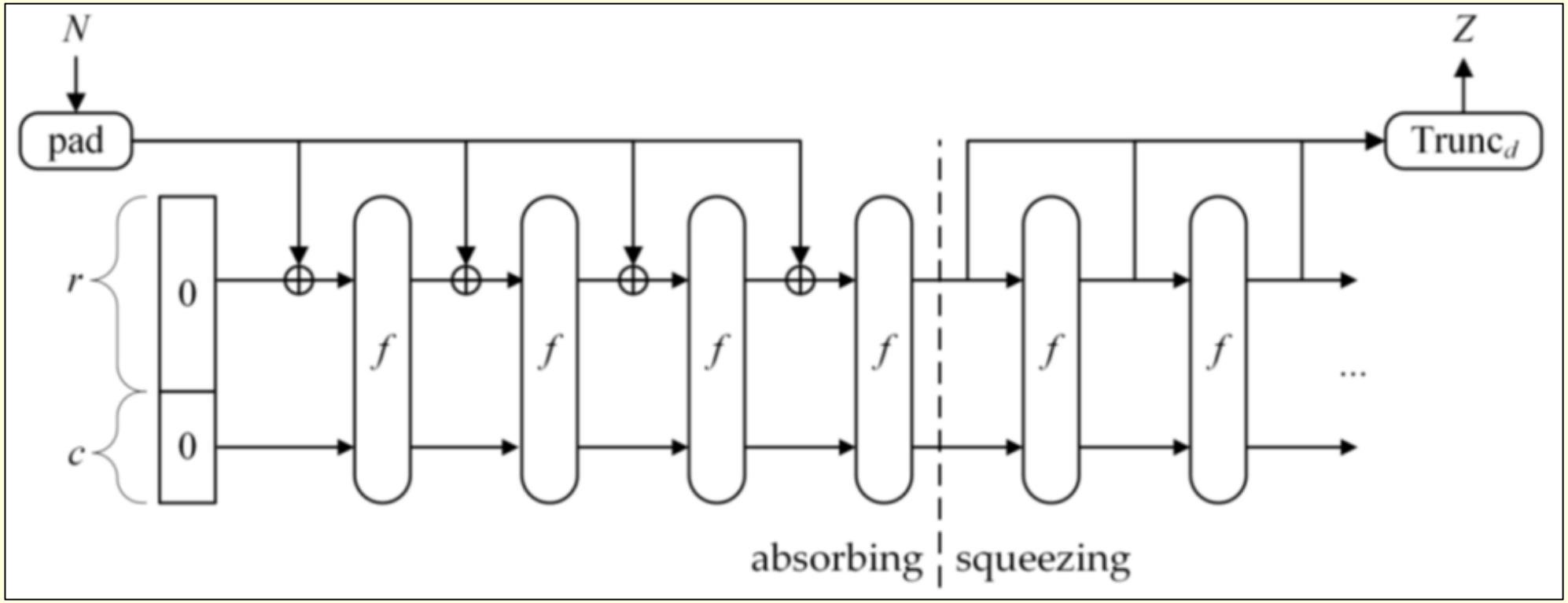
Конструкция губки

Конструкция состоит из следующих компонент:

1. Лежащая в основе функция для строк фиксированной длины, обозначаемая f
2. Параметр, называемый rate, обозначаемый r
3. Правило дополнения, обозначаемой pad

Функция называется губкой и обозначается SPONGE [f, pad, r]. Функция губки получает два входных значения: строку битов, обозначаемую N, и длину в битах, обозначаемую d, выходная строка SPONGE [f, pad, r](N, d).

Хеш-функции SHA-3



Функция f отображает строку фиксированной длины, обозначаемую b, в строку той же самой длины.

Хеш-функции SHA-3

Настройки

- Исходный алгоритм Keccak имеет множество настраиваемых параметров с целью обеспечения оптимального соотношения криптостойкости и быстродействия для определённого применения алгоритма на определённой платформе. Настраиваемыми величинами являются: **размер блока данных**, **размер состояния алгоритма**, **количество раундов в функции *f()*** и другие.
- На протяжения конкурса хеширования NIST участники имели право настраивать свои алгоритмы для решения возникших проблем. Так, были внесены некоторые изменения в Keccak: количество раундов было увеличено с 18 до 24 с целью увеличения запаса безопасности.
- Авторы Keccak учредили ряд призов за достижения в криптоанализе данного алгоритма.

Хеш-функции SHA-3

- Версия алгоритма, принятая в качестве окончательного стандарта SHA-3, имеет несколько незначительных отличий от исходного предложения Keccak на конкурс. В частности, были ограничены некоторые параметры (отброшены медленные режимы c=768 и c=1024), в том числе для увеличения производительности. Также в стандарте были введены «функции с удлиняемым результатом» (XOF, Extendable Output Functions) SHAKE128 и SHAKE256, для чего хешируемое сообщение стало необходимо дополнять «суффиксом» из 2 или 4 бит, в зависимости от типа функции.

Хеш-функции SHA-3

Функция	Формула
SHA3-224(*M*)	Keccak[448](*M*||01, 224)
SHA3-256(*M*)	Keccak[512](*M*||01, 256)
SHA3-384(*M*)	Keccak[768](*M*||01, 384)
SHA3-512(*M*)	Keccak[1024](*M*||01, 512)
SHAKE128(*M*, *d*)	Keccak[256](*M*||1111, *d*)
SHAKE256(*M*, *d*)	Keccak[512](*M*||1111, *d*)

Хеш-функции SHA-3

Производные функции

- В декабре 2016 года Национальный институт стандартов и технологий США опубликовал новый документ, NIST SP.800-185, описывающий производные функции на основе SHA-3:

cSHAKE128(*X*, *L*, *N*, *S*)	Параметризованная версия SHAKE
cSHAKE256(*X*, *L*, *N*, *S*)	
KMAC128(*K*, *X*, *L*, *S*)	Имитовставка на основе Keccak
KMAC256(*K*, *X*, *L*, *S*)	
KMACXOF128(*K*, *X*, *L*, *S*)	
KMACXOF256(*K*, *X*, *L*, *S*)	

Хеш-функции SHA-3

TupleHash128(*X*, *L*, *S*)	Хеширование кортежа строк
TupleHash256(*X*, *L*, *S*)	
TupleHashXOF128(*X*, *L*, *S*)	
TupleHashXOF256(*X*, *L*, *S*)	
ParallelHash128(*X*, *B*, *L*, *S*)	Параллелизуемая хеш-функция на основе Keccak
ParallelHash256(*X*, *B*, *L*, *S*)	
ParallelHashXOF128(*X*, *B*, *L*, *S*)	
ParallelHashXOF256(*X*, *B*, *L*, *S*)	

Хеш-функции ГОСТ 3411-94, 2012, 2018

- Алгоритм ГОСТ 3411 является отечественным стандартом на хеш-функции. Его структура довольно сильно отличается от структуры алгоритмов SHA-1,2 или MD5, в основе которых лежит алгоритм MD4.
- Длина хеш-кода, создаваемого алгоритмом ГОСТ 3411, равна 256 битам. Алгоритм разбивает сообщение на блоки, длина которых также равна 256 битам. Кроме того, алгоритм имеет параметр, который называется стартовый вектор хеширования **H** – произвольное фиксированное значение длиной также 256 бит.
- Сообщение обрабатывается справа налево блоками по 256 бит.
- Каждый блок сообщения обрабатывается по следующему алгоритму.

1. Генерация четырех ключей длиной 256 бит каждый.
2. Шифрование 64-битных значений промежуточного хеш-кода **H** на ключах $\mathbf{K_i}$, **(i = 1,2,3,4)** с использованием алгоритма ГОСТ 28147 в режиме простой замены.
3. Перемешивание результата шифрования.

Код аутентификации сообщения

- Рассмотрим обеспечение целостности сообщений с использованием общего секрета. Напомним, что обеспечение целостности сообщения – это невозможность изменения сообщения так, чтобы получатель этого не обнаружил. Под аутентификацией понимается подтверждение того, что информация получена от законного источника, и получателем является тот, кто нужно. Один из способов обеспечения целостности – это вычисление MAC (Message Authentication Code). В данном случае под MAC понимается некоторый аутентификатор, являющийся определенным способом вычисленным блоком данных, с помощью которого можно проверить целостность сообщения.
  - MAC вычисляется в тот момент, когда известно, что сообщение корректно. После этого MAC присоединяется к сообщению и передается вместе с ним получателю. Получатель вычисляет MAC, используя тот же самый секретный ключ, и сравнивает вычисленное значение с полученным. Если эти значения совпадают, то с большой долей вероятности можно считать, что при пересылке изменения сообщения не произошло.

$$MAC = C_K(M)$$

Код аутентификации сообщения

- Функция вычисления MAC должна обладать следующими свойствами:

1. Должно быть вычислительно трудно, зная M и $C_K(M)$, найти сообщение M', такое, что $C_K(M) = C_K(M')$.
2. Значения $C_K(M)$ должны быть равномерно распределенными в том смысле, что для любых сообщений M и M' вероятность того, что $C_K(M) = C_K(M')$, должна быть равна 2^{-n}, где n – длина значения MAC.

Стандарт HMAC

- Использование хеш-функции для получения MAC состоит в том, чтобы определенным образом добавить секретное значение к сообщению, которое подается на вход хеш-функции. Такой алгоритм носит название HMAC, и он описан в RFC 2104.
- При разработке алгоритма HMAC преследовались следующие цели:

1. Возможность использовать без модификаций уже имеющиеся хеш-функции;
2. Возможность легкой замены встроенных хеш-функций на более быстрые или более стойкие.
3. Сохранение скорости работы алгоритма, близкой к скорости работы соответствующей хеш-функции.
4. Возможность применения ключей и простота работы с ними.

Стандарт HMAC

- В алгоритме HMAC хеш-функция представляет собой «черный ящик». Это, во-первых, позволяет использовать существующие реализации хеш-функций, а во-вторых, обеспечивает легкую замену существующей хеш-функции на новую.
- Введем следующие обозначения:

  H – встроенная хеш-функция.

  b – длина блока используемой хеш-функции.

  n – длина хеш-кода.

  K – секретный ключ. К этому ключу слева добавляют нули, чтобы получить **b**-битовый ключ K^+.

- Вводится два вспомогательных значения:

Ipad - значение **'00110110'**, повторенное **b/8** раз.

Opad – значение **'01011010'**, повторенное **b/8** раз.

- Далее HMAC вычисляется следующим образом:

$$\mathbf{HMAC = H\ ((K^+ \oplus Opad)\ ||\ H\ ((K^+ \oplus Ipad)\ ||\ M))}$$

Стандарт Poly1305

- Poly1305 является одноразовым аутентификатором, разработанным D. J. Bernstein. Poly1305 **получает 32-байтный одноразовый ключ и сообщение и создает 16-байтный тег (128 бит).** Данный тег используется для аутентификации сообщения.
- Первоначально Poly1305 разрабатывался для использования совместно с AES для создания кода аутентификации сообщения (**MAC**). Для MAC-функции требуется **128-битный ключ AES**, **128-битный дополнительный ключ** и **128-битный (несекретный) nonce**. AES в данном случае используется для шифрования nonce, чтобы получить уникальную (и секретную) 128-битную строку, но при этом не обязательно должен использоваться AES. Можно заменить AES произвольной функцией с ключом и произвольным набором nonce, из которых получаются 16-байтные строки.

Стандарт Poly1305

$$Poly1305_r (M, AES_k (n))$$

M – сообщение произвольной длины

Исходный стандарт, Poly1305-AES, использует функцию шифрования AES в качестве источника псевдослучайности, вычисляет 128-битный (16 байт) аутентификатор сообщения. Для вычисления MAC используется 128-битный **ключ AES**, 128-битный (16-байтный) дополнительный **ключ r** и 128-битный **nonce**, который должен быть уникальным среди всех сообщений, аутентифицированных с помощью одного и того же ключа.

Стандарт Poly1305

Сообщение **c** разбивается на **16-байтовые блоки**, которые становятся коэффициентами многочлена, вычисляемого в **r (дополнительный 16-байтный ключ)**, по модулю простого числа 2^{130}-5.

$$r = r[0] + 2^8\, r[1] + \ldots + 2^{120}\, r[15]$$

Код аутентификации представляет собой сумму этого полинома плюс значение, являющееся результатом шифрования nonce алгоритмом AES.

$$((c_1 r^q + c_2 r^{q-1} + \ldots + c_q r^1) \bmod (2^{130} - 5) + AES_k(n)) \bmod 2^{128}$$

q – длина сообщения, деленная на 16.

В Poly1305 может использоваться Salsa20 вместо AES, а в TLS и SSH ChaCha20. Google выбрал Poly1305 вместе с симметричным шифром ChaCha20 в качестве замены RC4 в TLS. Использование ChaCha20 вместе с Poly1305 стандартизировано в RFC 7905.

Стандарт AEAD

Конструкция AEAD (Authenticated Encryption with Associated Data)

Rfc 5116, 2008г.

- AEAD_CHACHA20_POLY1305 является алгоритмом аутентифицированного шифрования с дополнительными данными. Входными значениями в AEAD_CHACHA20_POLY1305 являются:
  - **256-битный ключ**
  - **96-битный nonce**, который отличается для каждого вызова с одним и тем же ключом
  - Незашифрованный текст (**plaintext**) произвольной длины
  - Дополнительные аутентифицированные данные (**AAD**) произвольной длины

Стандарт AEAD

Вызывается функция **Poly1305** с ключом, входное сообщение является конкатенацией следующего:

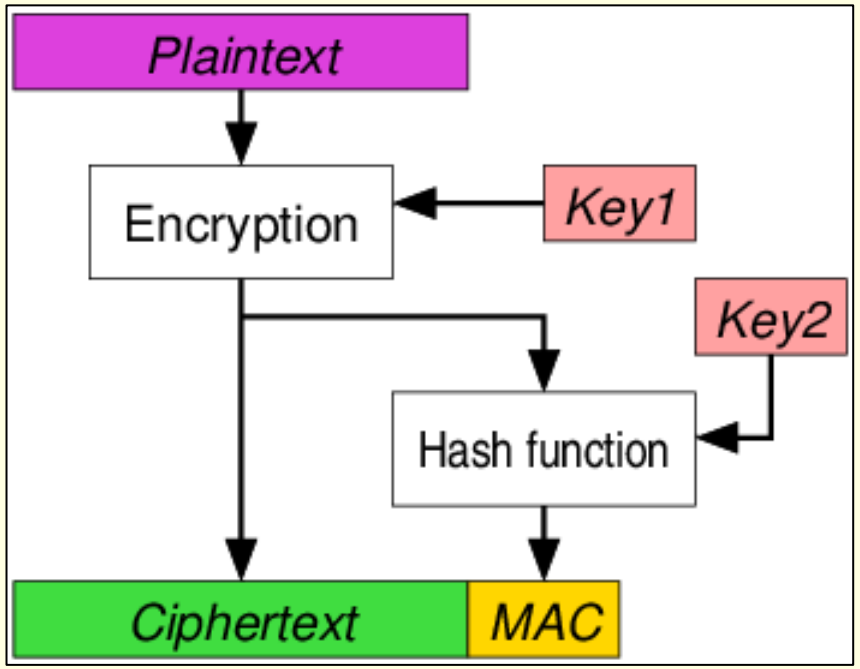
- AAD – дополнительные данные произвольной длины
- Padding 1 – добавление, чтобы в результате общая длина стала кратной 16. Если длина AAD уже была кратна 16 байтам, данное поле имеет нулевую длину.
- Зашифрованный текст
- Padding 2 – добавление максимум 15 нулевых байтов, в результате общая длина становится кратной 16. Если длина зашифрованного текста уже была кратна 16 байтам, данное поле имеет нулевую длину.
- Длина дополнительных данных в октетах (как 64-битное little-endian целое).
- Длина зашифрованного текста в октетах (как 64-битное little-endian целое).

Стандарт AEAD

- **Выходное значение AEAD** является конкатенацией следующего:
  - **Зашифрованное сообщение** той же самой длины, что и plaintext.
  - 128-битное выходное значение функции **Poly1305**.
- Расшифрование аналогично со следующими различиями:
  - Меняются местами роли ciphertext и plaintext, т.е. функция шифрования ChaCha20 применяется к ciphertext, создавая plaintext.
  - Функция Poly1305 выполняется над AAD и ciphertext, не над plaintext.
  - Вычисленный тег побитово сравнивается с полученным тегом. Сообщение считается аутентифицированным тогда и только тогда, когда теги совпали.

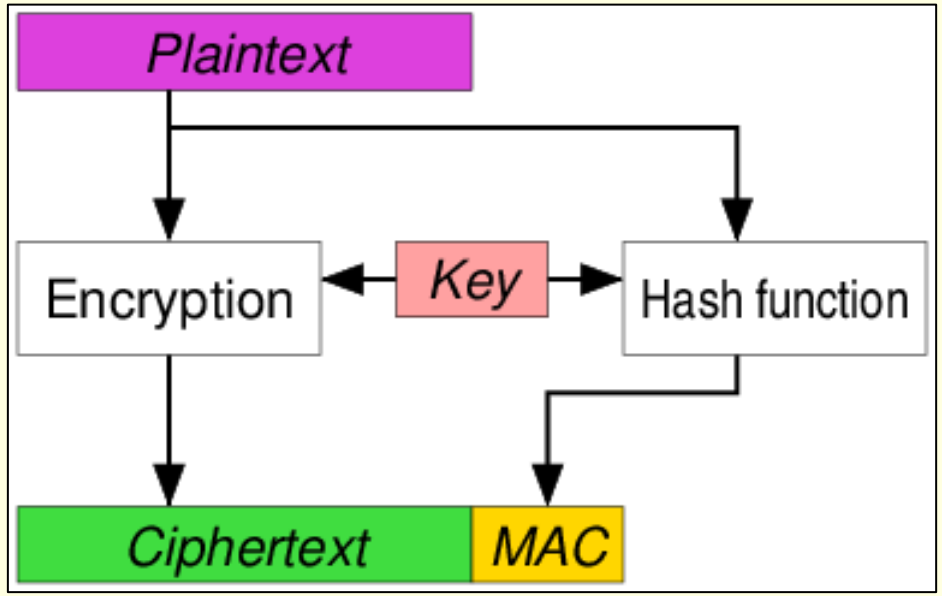
Варианты аутентифицированного шифрования

- Encrypt-then-MAC (EtM)



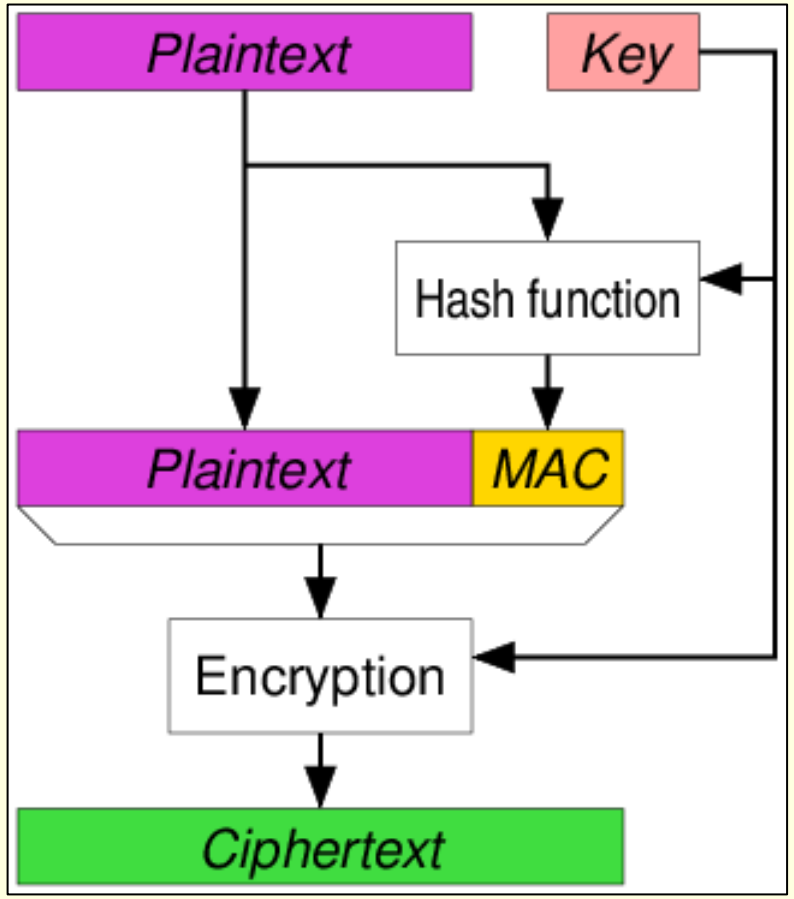
Варианты аутентифицированного шифрования

- Encrypt-and-MAC (E&M)



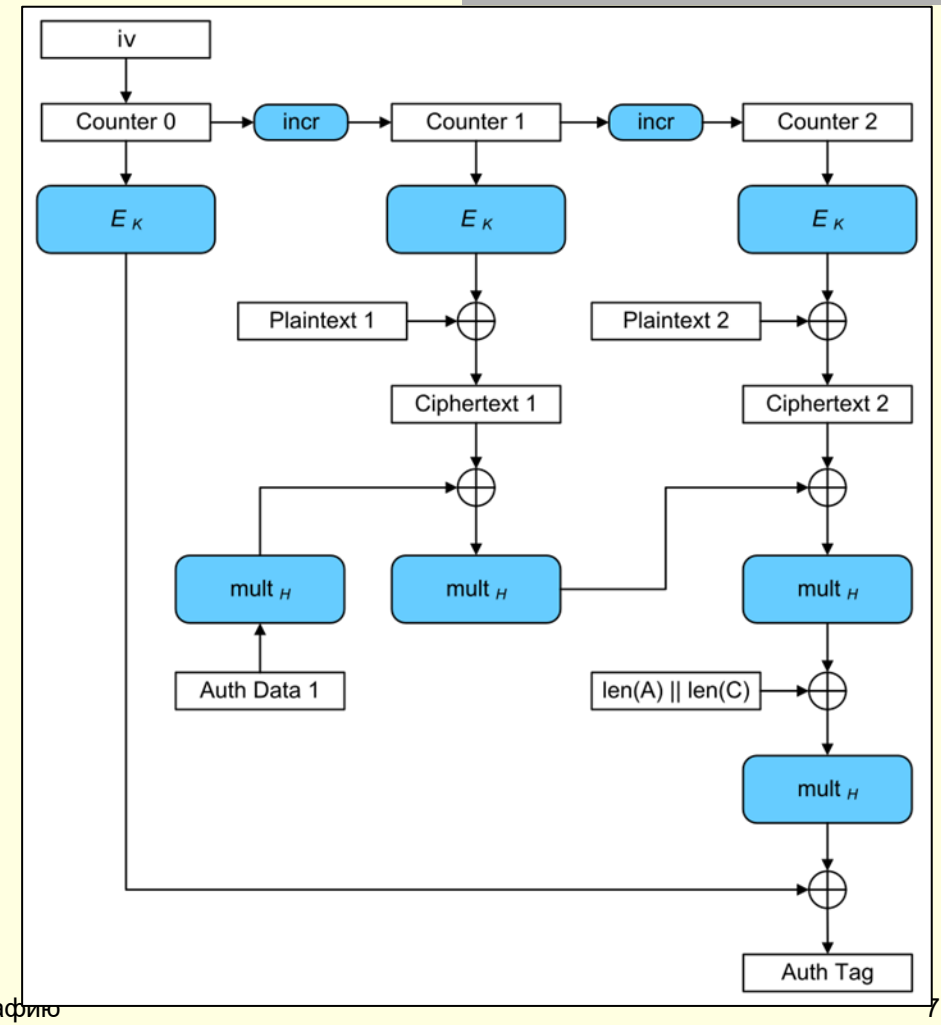
Варианты аутентифицированного шифрования

- MAC-then-Encrypt (MtE)



Режим Galois/Counter Mode (GCM) и AEAD

- Galois/Counter Mode (счётчик с аутентификацией Галуа) — более безопасная модификация CTR, предоставляющее аутентифицированное шифрование с присоединёнными данными (AEAD-режим блочного шифрования).



Режим Galois/Counter Mode (GCM) и AEAD

- В обычном режиме шифрования CTR входные блоки нумеруются последовательно, номер блока шифруется блочным алгоритмом E (например, AES). Выход функции шифрования используется в операции XOR (исключающее или) с открытым текстом для получения шифротекста. Как и для других режимов на базе счётчиков, схема представляет собой потоковый шифр, поэтому обязательным является использование уникального вектора инициализации для каждого шифруемого потока данных.
- В GCM используется функция Галуа GHASH (H, A, C) («Mult»), которая комбинирует блоки шифротекста и код аутентификации, чтобы получить тег аутентификации. На вход функции подается ключ хеширования H, являющийся результатом шифрования 128 нулевых битов на ключе K, т.е. $\mathbf{H = E\,(K,\ 0^{128})}$. Тег аутентификации используется для проверки целостности сообщения. Получателю передаются: вектор инициализации IV, блоки шифротекста, и код аутентификации (16 байтов). По своим свойствам режим GCM (GMAC) похож на HMAC.