Прикладная Криптография: Симметричные криптосистемы Хэш-функции

Макаров Артём

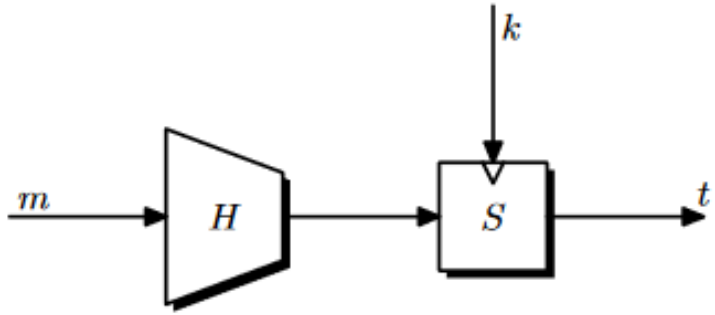
МИФИ 2019

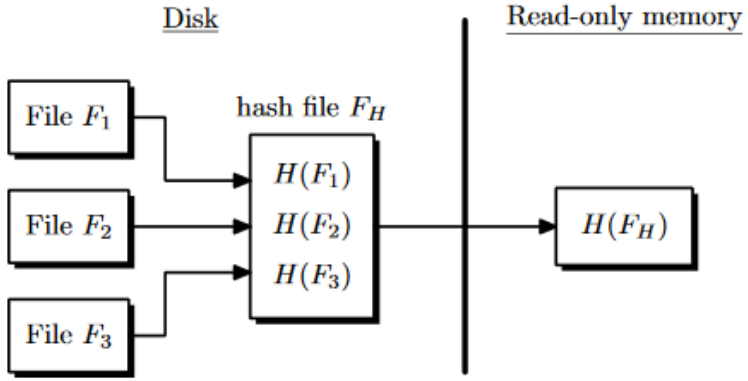
Целостность сообщений

- Рассмотрим бесключевые хэш-функции
- Задача – получить функцию, для которой нахождение коллизии является сложной задачей
- Хотим построить такую функцию $H: M \to T: m_0, m_1 \in M, |T| \leq |M|$
- Коллизия $(m_0, m_1) \in M^2: m_0 \neq m_1, H(m_1) = H(m_2)$

Применение хэш-функций

- Расширение множества значений криптографических примитивов, обеспечивающих аутентичность и целостность (hash-then-mac, hash-then-sign). Возможно вычислить **MAC** или **цифровую подпись** для сообщения (**произвольной длины**), подписывая хэш от него, и используя только один вызов процедуры подписи на одном блоке.
- Обеспечение целостности файлов в файловой системе. Пусть существует n часто изменяющихся файлов. Хотим проверить их целостность (что они не были модифицированы злоумышленником или вирусом). Используем read-only память для хранения хэш-значения от этих файлов. Для проверки достаточно повторно пересчитать это значение и сверить с хранимым.

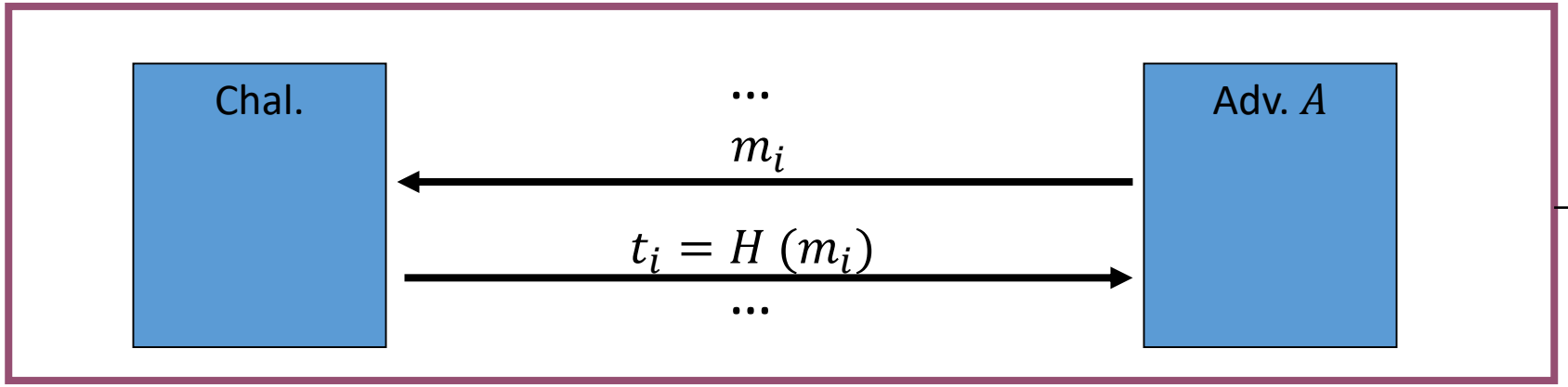




Определение хэш-функции

Хэш-функция H на (H, M), $H: M \to T, |M| > |T|$ - эффективно вычислима.

Игра на стойкость к коллизиям. Пусть противнику дан оракул хэш-функции H на (M, T) (доступ к ней через претендента). Задача противника – получить пару сообщений $(m_0, m_1) \in M^2: m_0 \neq m_1, H(m_0) = H(m_1)$.

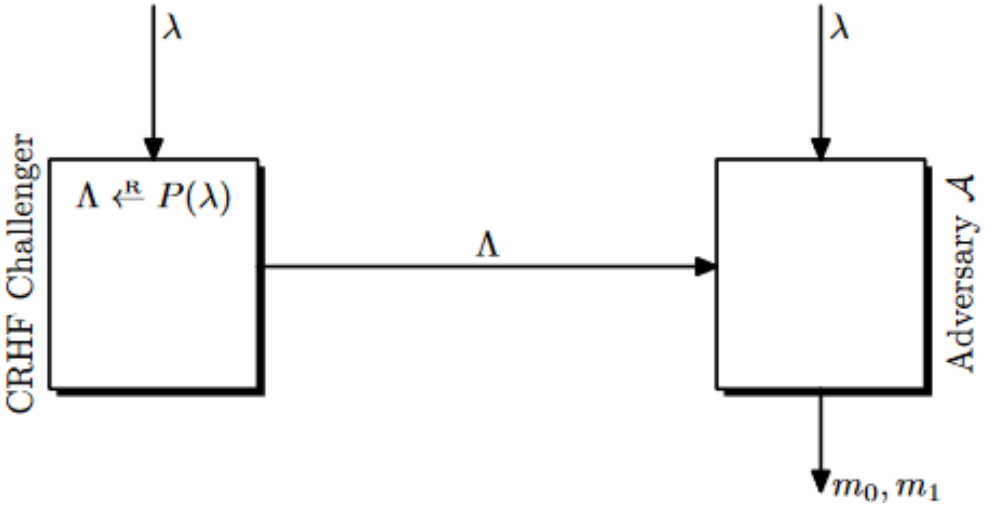


Обозначим преимуществом противника A против H через $CR_{adv}[A, H] = \Pr[A \text{ победил в игре}]$

Определение хэш-функции

Функция H на (H, M) называется **стойкой к коллизиям хэш-функцией**, если $\forall A$ величина $CR_{adv}[A, H] \leq \epsilon$, где ϵ – пренебрежимо малая.

Формально говоря, хэш-функция может быть параметризована некоторым системным параметром λ, определяющим выбор конкретной хэш-функции из семейства хэш-функций. Однако, предполагается что противник тоже знает этот системный параметр.



Построение MAC для произвольных сообщений

- Пусть $I = (S, V)$ – стойкий MAC для на (T_H, T), H – стойкая к коллизиям хэш-функция на (M, T_H), $|M| > |T_H|, |M| > |T|$.
- Построим новый MAC $I' = (S', V')$:

$$S'(k, m) = S\big(k, H(m)\big)$$
$$V'(k, m) = V(k, H(m))$$

Теорема 11.1. MAC I' описанный выше – стойкий MAC, причём $\forall A$ в игре на стойкость MAC $\exists B_I$ в игре на стойкость MAC и B_H в игре на стойкость к коллизиям, такой что

$$MAC_{adv}[A, I'] \leq MAC_{adv}[B_I, I] + CR_{adv}[B_H, H]$$

▻ идея доказательства – если противник выдал новую пару сообщение-MAC то она либо сломал MAC, либо нашёл коллизию◅

Атаки на основе парадокса дней рождений

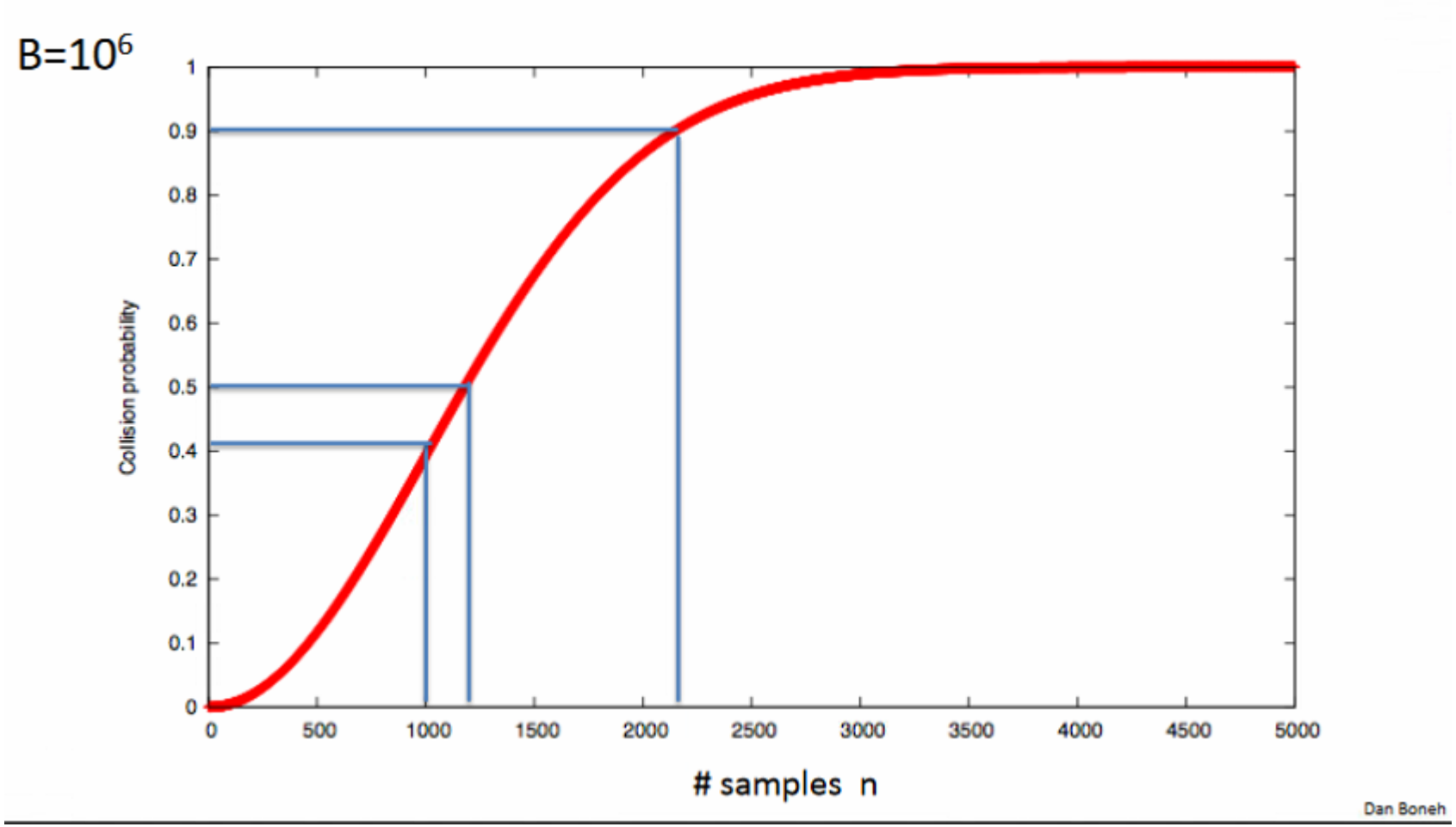
Пусть $H: M \to T$, - хэш-функция. $T = \{0,1\}^n$

Алгоритм перебора для нахождения коллизии:

- Выбрать $2^{n/2}$ случайных сообщений из M
- Вычислить $t_i = H(m_i)$
- Найти коллизию $t_i = t_j, i \neq j$

Вероятность успешного завершения алгоритма = ½ (из за парадокса дней рождений). Сложность атаки $\sim 2^{n/2} = \sqrt{|T|}$

Следовательно, чем меньше область определений хэш-функции, тем проще атаковать хэш-функцию используя алгоритм выше.


B=10^6
Collision probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
samples n
Dan Boneh

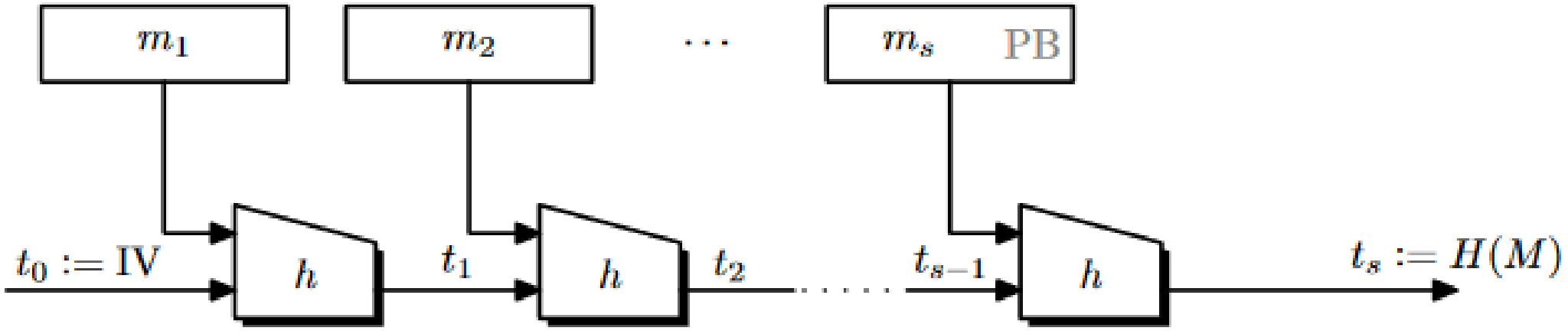
Парадигма Меркла-Дамгарда

Большинство современных хэш-функций стоится по итеративному принципу. Сначала описывается некоторая хэш-функция для сообщений малой длины, которая затем итеративно используется для хэша для сообщений произвольной длины.

Пусть $h: X \times Y \to X$ – хэш-функция. Пусть $Y = \{0,1\}^l$. Функцией Меркла-Дамгарда H_{MD} на основе хэш-функции h называется следующий алгоритм:

- $M' \leftarrow M||PB$ # дополнение до длины, кратной l
- $M' = m_1|| \dots ||m_s$, где $m_i \in \{0,1\}^l$
- $t_0 \leftarrow IV \in x$
- For $i = 1..s$ do:

  $$t_i \leftarrow h(t_{i-1}, m_i)$$
- Return t_s


m_1
m_2
$\cdots$
m_s PB
$t_0 := \mathrm{IV}$
h
t_1
h
t_2
t_{s-1}
h
$t_s := H(M)$

Парадигма Меркла-Дамгарда

Функция h - называется **функцией сжатия**.

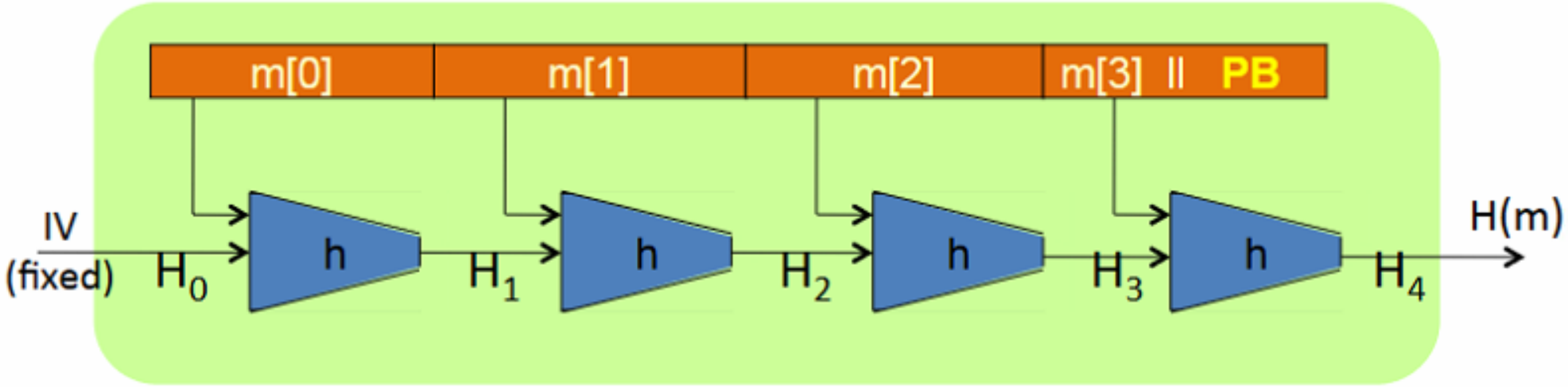
IV – некоторая **константа**, называемая **инициализирующим значением**.

$m_1, .., m_s$ - блоки сообщений

PB – **блок дополнения**. Формат блока дополнения $PB = 100 \ldots 00||\{s\}$, где $\{s\}$ – число блоков в сообщении в двоичном представлении. Обычно $\{s\}$ составляет 64 бита.

Для описания хэш-функции необходимо задать **функцию сжатия**, **инициализирующее значение** и **дополнение**.

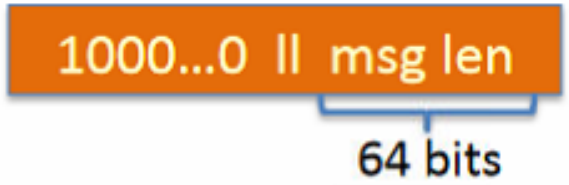
Парадигма Меркла-Дамгарда



Given $h: T \times X \longrightarrow T$ (compression function)

we obtain $H: X^{\leq L} \longrightarrow T$. H_i - chaining variables

PB: padding block



If no space for PB
add another block

Стойкость схемы Меркла-Дамгарда

Теорема 11.2. Пусть L – полиномиально ограниченная величина и h - стойкая к коллизиям хэш-функция на $(X \times Y, X)$. Тогда хэш-функция Меркла-Дамгарда H_{MD}, построенная на основе h и определённая на $(\{0,1\}^L, X)$ – стойкая к коллизиям, причём $\forall A$ в игре против H_{MD} $\exists B$ в игре против h:

$$CR_{adv}[A, H_{MD}] = CR[B, h]$$

⊳ без доказательства⊲

Построение функций сжатия

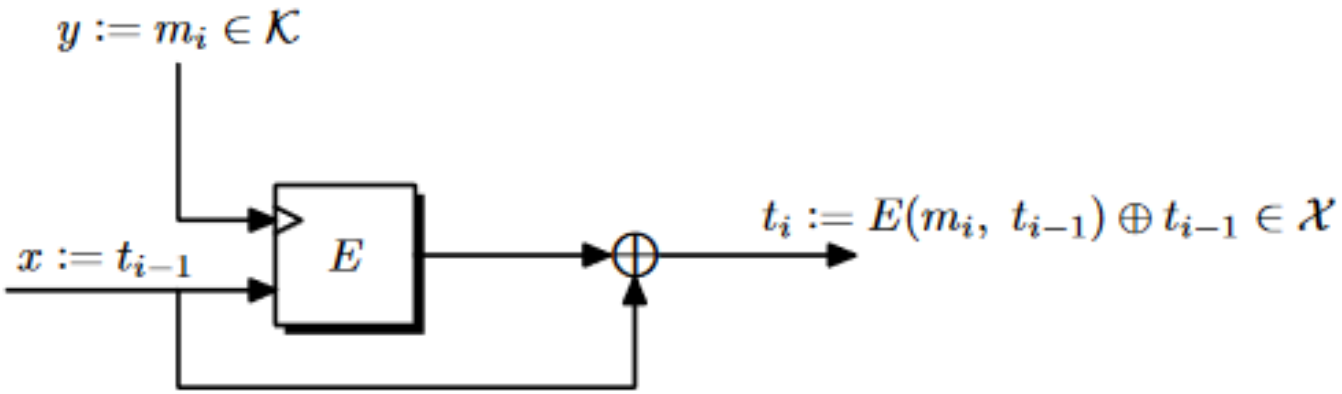
Рассмотрим метод Девиеса-Меера построения функций сжатия.

Пусть $\mathrm{E} = (E, D)$ блочный шифр на $(K, X), X = \{0,1\}^n$.

Введём функцию

$$h_{DM}(x, y) = E(y, x) \oplus x$$

на $(X \times K, X)$.



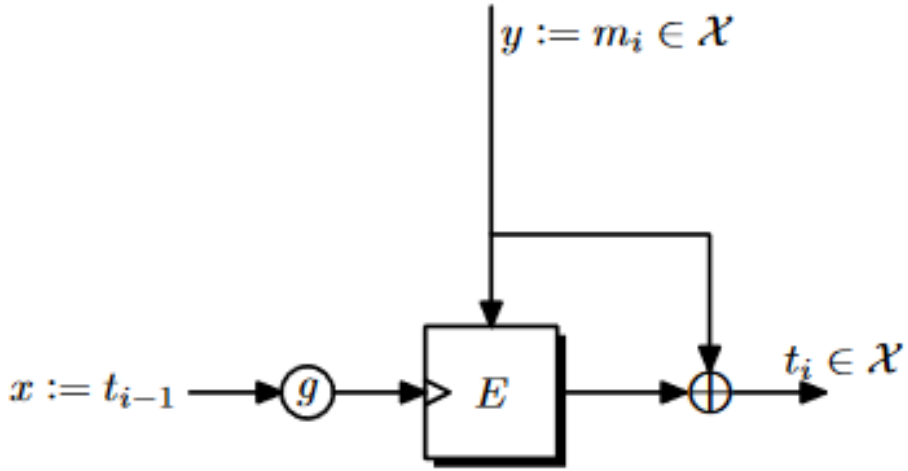
Построение функций сжатия

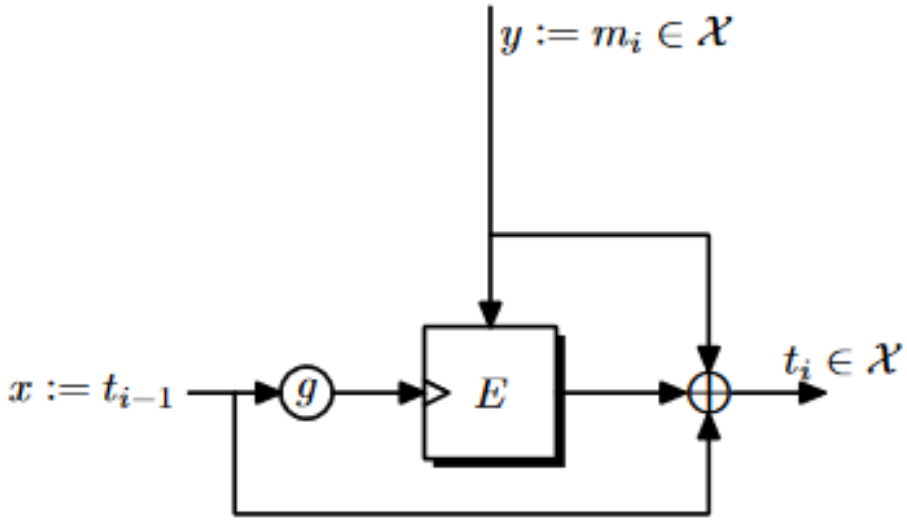
Блочные шифры часто оптимизируются при построении и реализации в предположении, что ключ блочного шифра будет использоваться для шифрования множества блоков. Постоянная смена ключа может значительно ухудшить производительность.

Следовательно, необходимо построить специальные блочные шифры, для которых частая смена ключа не окажет влияние на их произвольность.

Вариации Девиеса-Меера

$g: X \to K$ – некоторая заданная функция кодирования (некоторое преобразование).





SHA-1

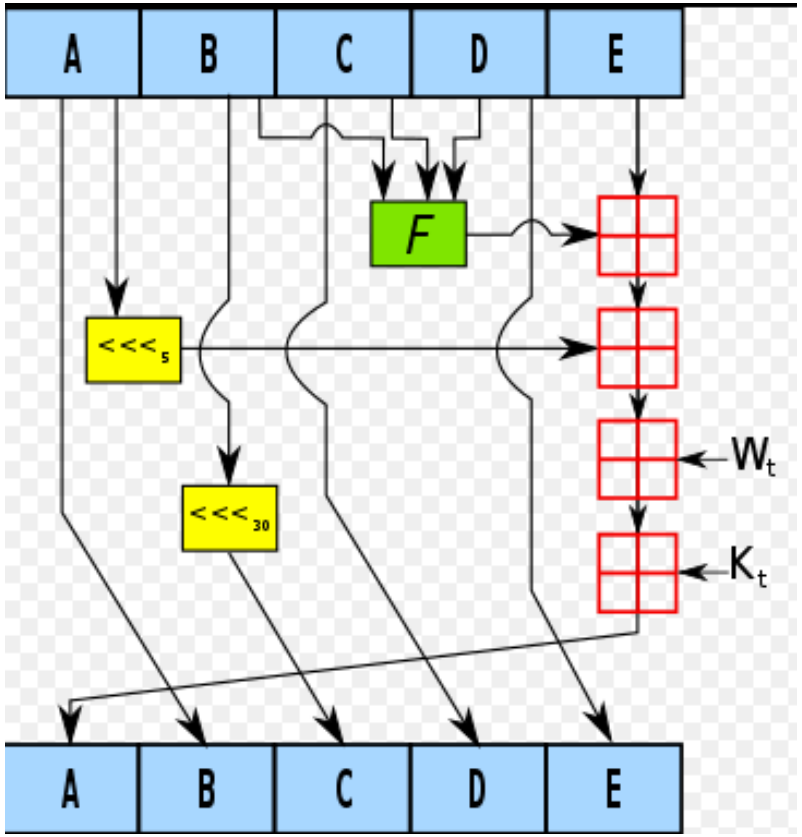
NIST 1993

Размер выхода – 160 бит

Построена с использованием парадигмы Меркла-Дамгарда

Являлась де-факто и де-юре стандартом (до сих пор используется во множестве legacy систем)

Сложность современной атаки - 2^{60}. Получена префиксная коллизия (т.е. добавление префикса к любым сообщением одинаковой длины даст одинаковый хэш)



A, B, C, D, E – 32 бита
F – нелинейная функция
W_t - слово (32 бита) полученное из сообщения
K_t - раундовая константа

SHA-2

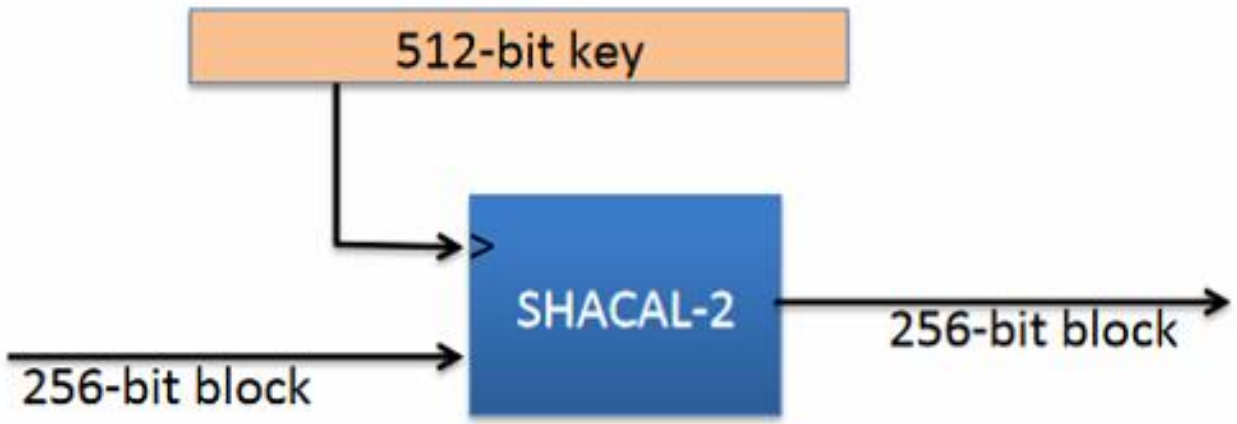
NIST 2002

Построена с использованием парадигмы Меркла-Дамгарда

Функция сжатия Девиеса-Меера

Блочный шифр – SHACAL-2

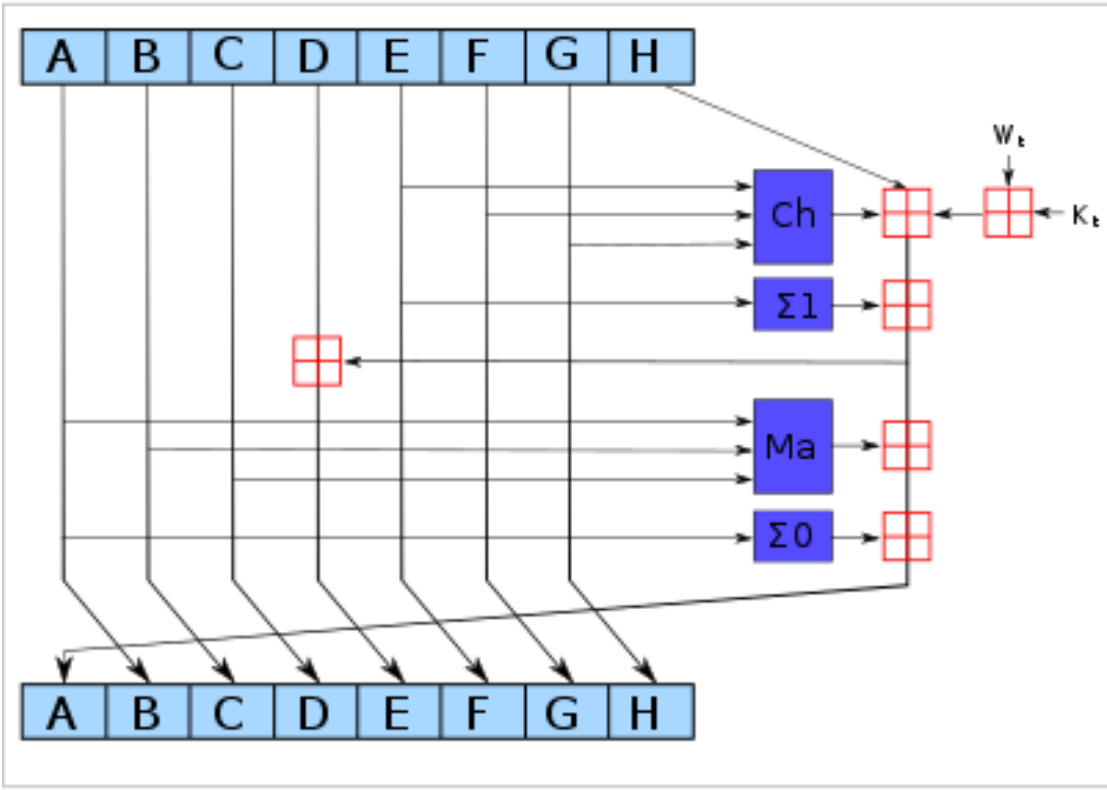
Современный стандарт хэш-функции



SHA-2

Размер выходы 256 или 512 бит

Атаки на полную схему не известны



One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.

The red ⊞ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

А ещё

- RIPEMD-160
- ГОСТ 34.11-94
- ГОСТ 34.11-2012 (Стрибог)
- MD-5 – СЛОМАН! (коллизии второго рода, хотя много где используется)

SHA-3

NIST 2015

Размер выхода – произвольный

Построена с использованием губчатой функции Keccak f1600 (f800)

Атаки на полную схему не известны

Стандарт на замену sha-2

Губчатая конструкция

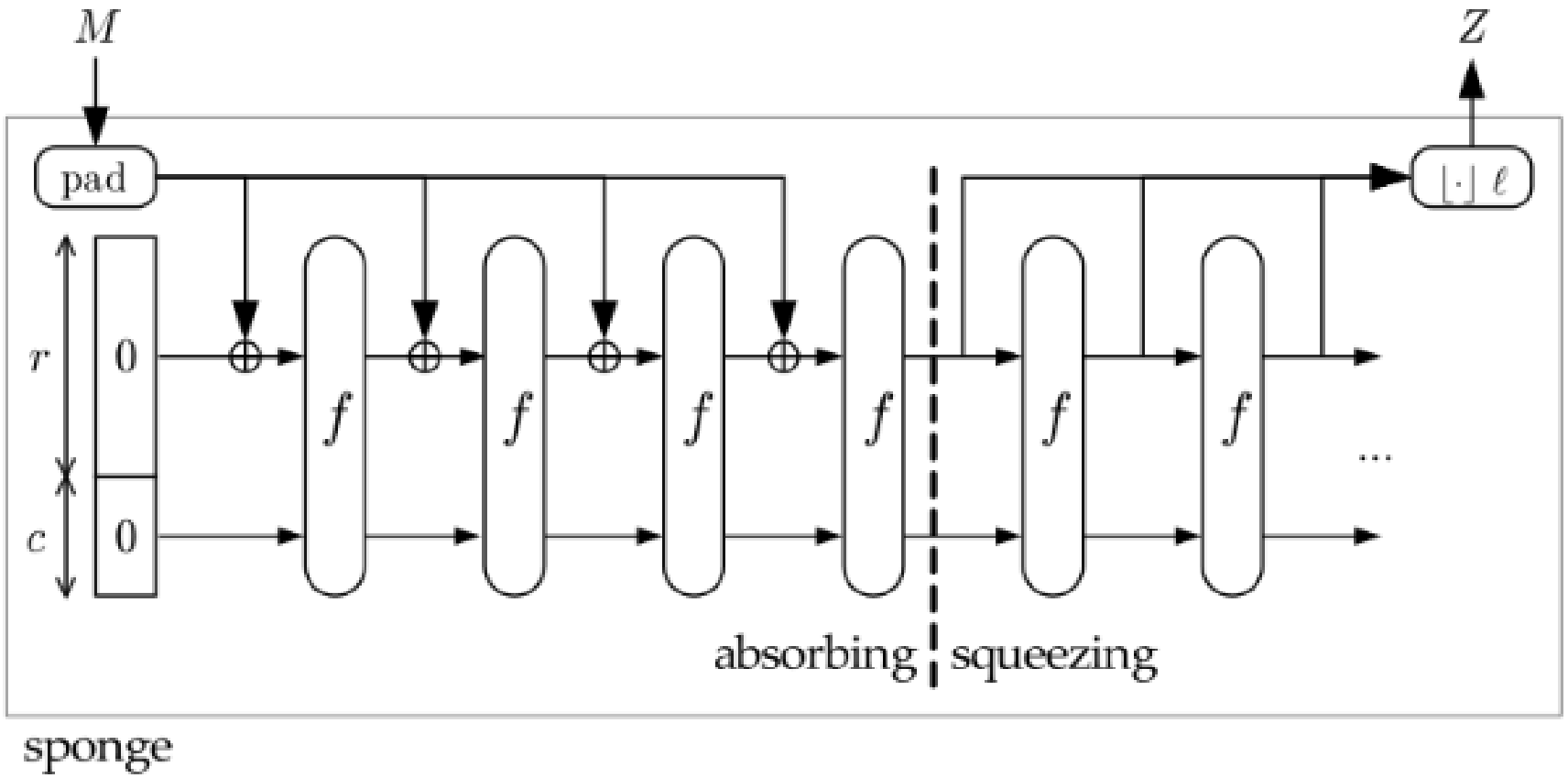
Основа губчатой конструкции – «волшебная» перестановка на некотором множестве.

Вводится понятие состояния – некоторого вектора, разделённого на 2 части – открытую и закрытую.

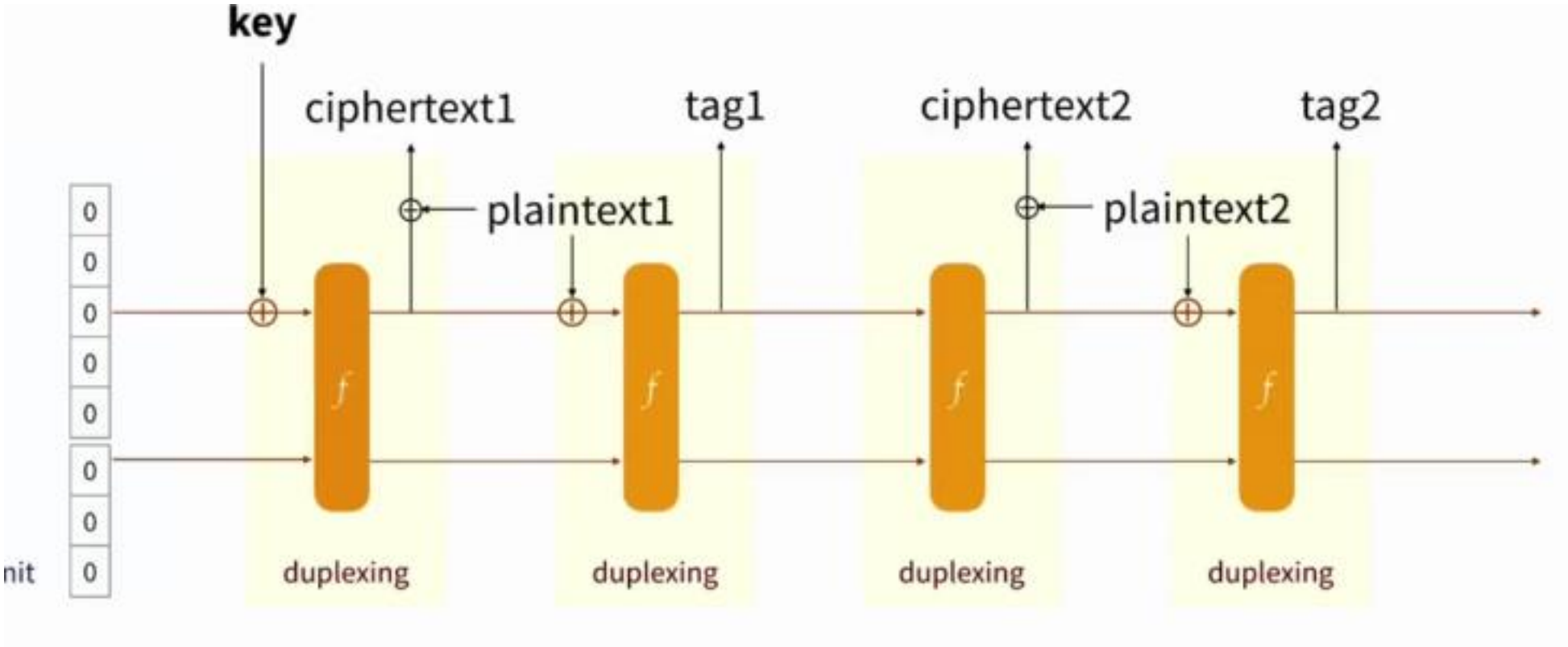
На каждом раунде открытая часть может изменяться входными данными или выдаваться в качестве входа, после чего вычисляется новое состояние с использованием перестановки.

Две основных операции – поглощение и «выжимание»

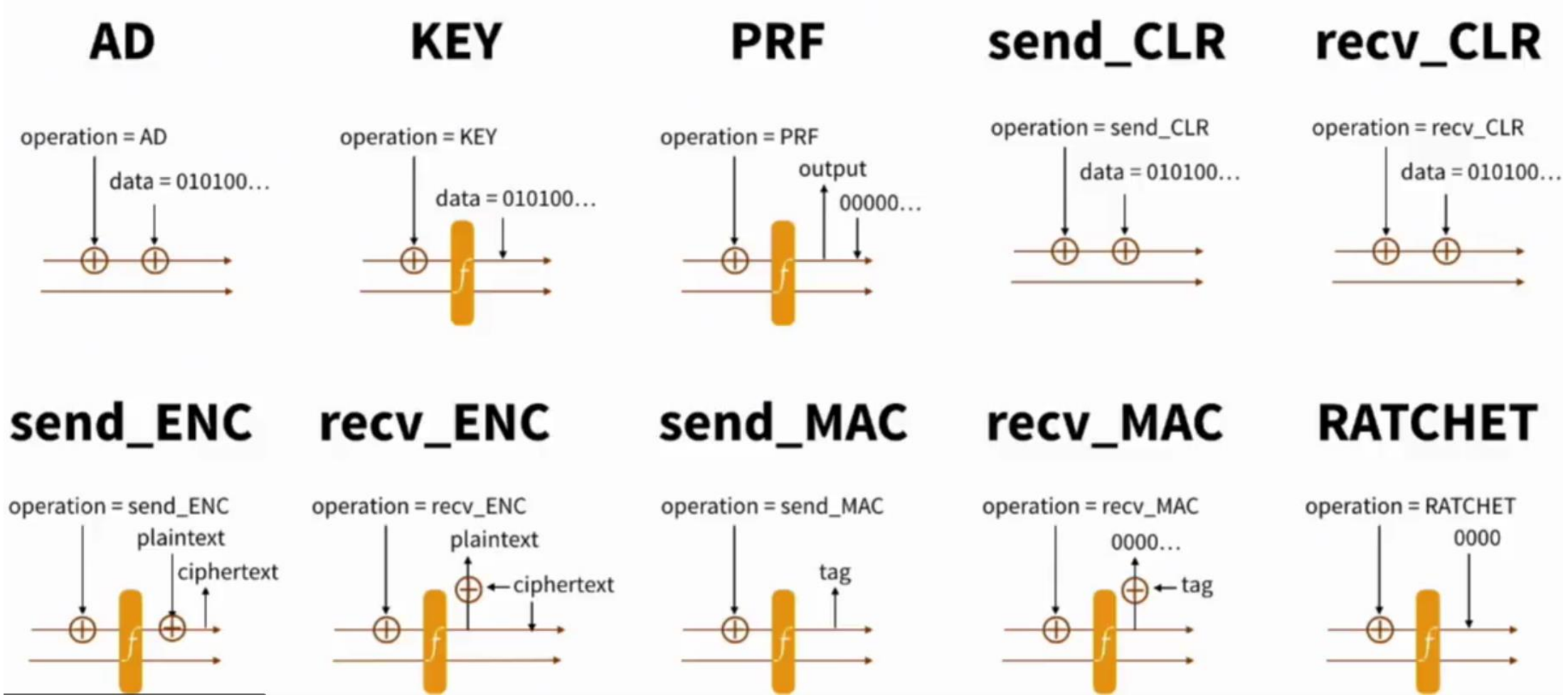
Губчатая конструкция (SHA-3)



Построение симметричной криптографии с использованием губчатой конструкции (Strobe)



Построение симметричной криптографии с использованием губчатой конструкции (Strobe)



Модели хэш-функций

До этого мы рассматривали стойкие хэш-функции, как функции **стойкие к коллизиям**. (**стойкие к коллизиям второго рода**)

Существуют и другие модели. Пусть H – хэш-функция на (M, T)

- H - **односторонняя хэш-функция**, если имея $t = H(m)$ для случайного $m \in M$ вычислительно сложно найти $m' \in M: H(m') = t$ (т.е. сложно обратить) (**preimage resistant)**
- H – **стойкая к коллизиям первого рода**, если имея случайное сообщение $m \in M$ сложно найти $m' \neq m: H(m) = H(m')$ (**2nd-preimage resistant**)
- H – **случайный оракул**, если оракул H реализует случайную функцию

Модели хэш-функций

Взаимосвязь моделей

Случайный оракул => стойкость к коллизиям второго рода => стойкость к коллизиям первого рода => односторонняя хэш-функция

Random oracle => collision resistance => 2nd-preimage resistant => one-way (preimage resistant)

Обратное вообще говоря не верно. Пример – SHA-1 сейчас считается стойкой односторонней хэш-функцией, но не стойкость к коллизиям второго рода.

Модели хэш-функций

	Русская терминология	Английская терминология	Отношение стойкости
1	Односторонняя хэш-функция	Preimage resistant, one-way (preimage resistant) (aka стойкость к нахождению прообраза)	=> 1
2	Стойкая к коллизиям первого рода	2nd-preimage resistant (aka стойкость к нахождению второго прообраза, когда один уже дан)	$\Rightarrow$ 12
3	Стойкие к коллизиям второго рода	Collision Resistant (aka стойкость к коллизиям)	$\Rightarrow$ 123
4	Случайный оракул (в старой терминологии не используется)	Random oracle	$\Rightarrow$ 1234

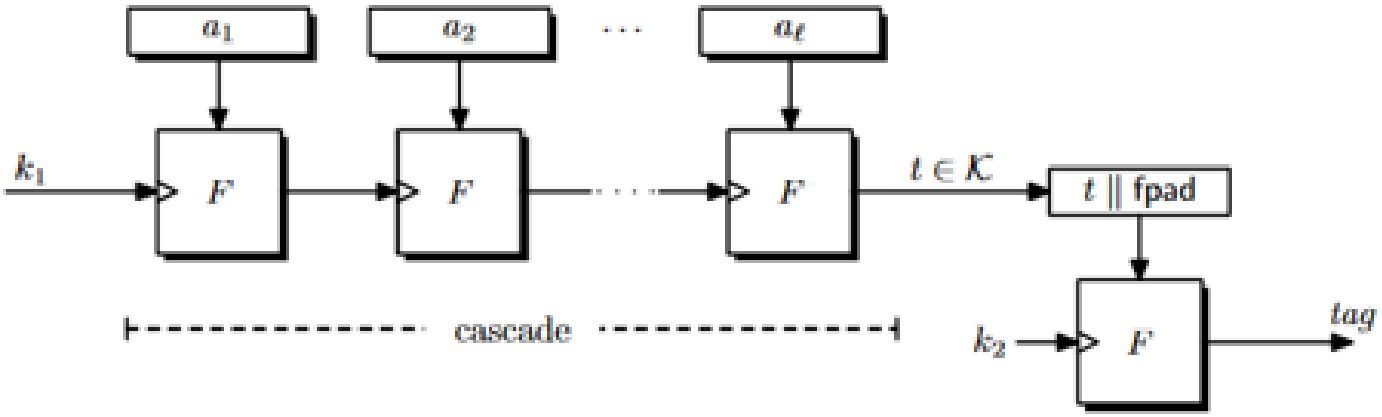
Модели хэш-функций

Т.к. хэш-функции широко распространены как криптографический примитив, стойкость системы часто сводится к стойкости хэш-функции в какой либо модели. При этом стараются использовать наиболее «слабую» модель хэш-функции, для обеспечения минимальных требований к хэш-функции и увеличения стойкости системы.

Некоторые доказательства для современных систем удалось провести только в модели случайного оракула (Random oracle model), в которой хэш-функции предполагаются случайными оракулами.

Модель без случайных оракулов называется стандартной (предполагая ограничение по времени и вычислительной мощи противника).

NMAC



- PRF F на $(K, M, K), K = \{0$
- $g(t) = t||$fpad, fpad – фиксированное дополнение, длины $n - k$ бит

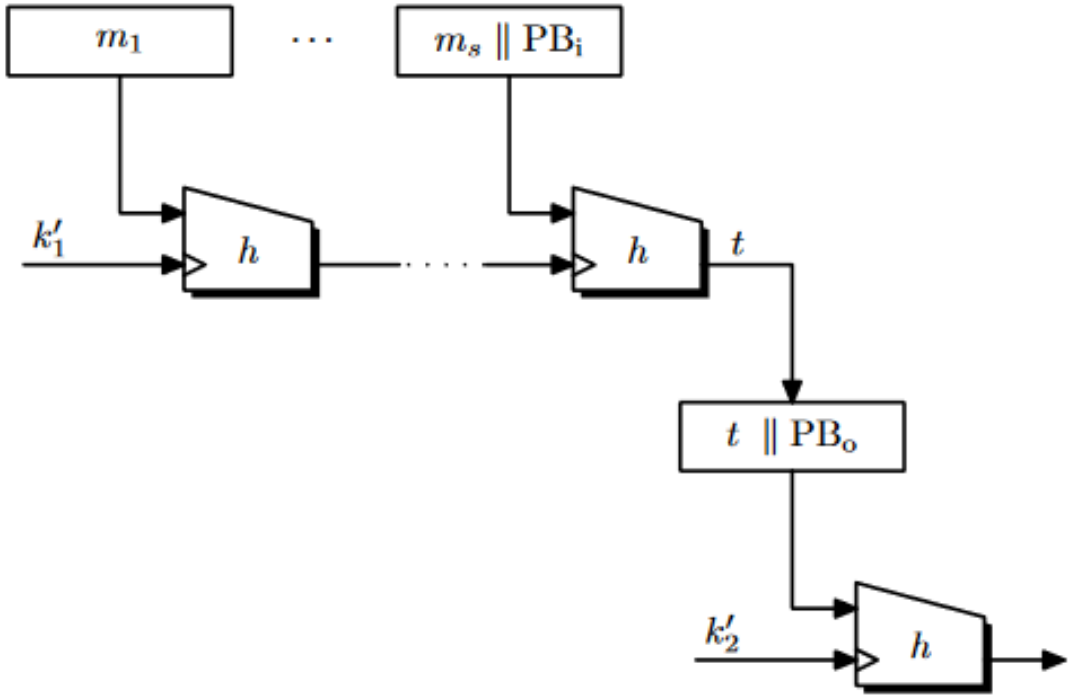
Теорема 10.3. $NMAC$, использующая PRF F стойкая PRF на $\left(K^2, X^{\leq l}, K\right)$:

$$PRF_{adv}[A, NMAC] \leq \left(Q(l+1)\right) * PRF_{adv}[B_1, F] + PRF_{adv}[B_2, F] + \frac{Q^2}{2|K|}$$

Рассмотрим как можно получить MAC на основе хэш-функции, используя похожую конструкцию.

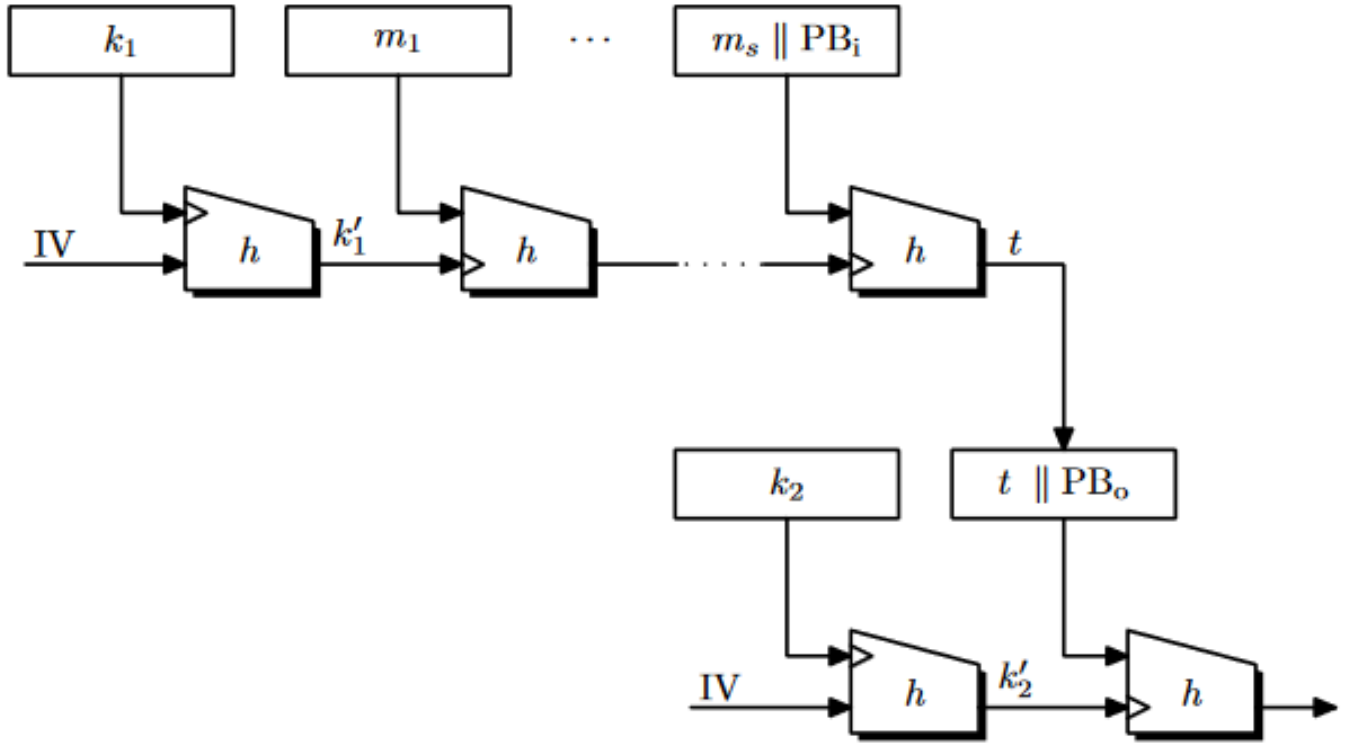
NMAC

- Заменим F на итеративную хэш-функцию хэш-функцию. Получим:



NMAC

Реализуем алгоритм получения ключей k_1' и k_2' с помощью IV и k_1 и k_2



HMAC

Уберём независимость ключей k_1 и k_2:

- $k_1 \leftarrow k \oplus ipad$
- $k_2 \leftarrow k \oplus opad$

Где

- $ipad = (0\text{x}36, 0\text{x}36, \ldots, 0\text{x}36)$
- $opad = (0\text{x}5\text{c}, 0\text{x}5\text{c}, \ldots, 0\text{x}5\text{c})$

Итого:

$$HMAC(k, m) = H(k \oplus opad \| H(k \oplus ipad, m))$$

Стойкость как PRF показана в модели стойкой PRF h при связанных ключах (related key attack PRF, RKA-PRF), что в свою очередь может быть доказано в модели идеального шифра (не вводили в данном курсе).

HMAC

- Де-факто интернет стандарт
- Не требует блочного шифра для реализации, основан на хэш-функции
- Используется во множестве протоколов
- Самый распространённых MAC
- Может быть построен с использованием произвольной хэш-функции (включая ГОСТ)
- В настоящий момент используется HMAC-SHA-256
- Лучше избегать использование HMAC-SHA-1, хотя в настоящий момент не известны практические атаки, существенно лучше перебора

Получение ключей

Пусть имеется единственный ключ SK, полученный из:

- Аппаратного датчика случайных чисел
- Протокола распределения ключей (передача или согласование ключа)
- Пароль пользователя

Хотим получить сессионные ключи из данного ключа.

Используются KDF – key derivation functions

Если источник ключей имеет равномерное распределение

$F: K \times X \to \{0,1\}^n$ - PRF

KDF(SK, CTX, L) :=

F(SK, (**CTX II 0**)) II F(SK, (**CTX II 1**)) II ⋯ II F(SK, (**CTX II L**))

CTX: контекст, строка, уникально представляющая приложение или назначению ключей (для независимой генерации различных ключей для различных приложений).

Если источник не имеет равномерное распределение

Напомним – PRF стойкая, только если ключи – случайные и равномерно распроданные.

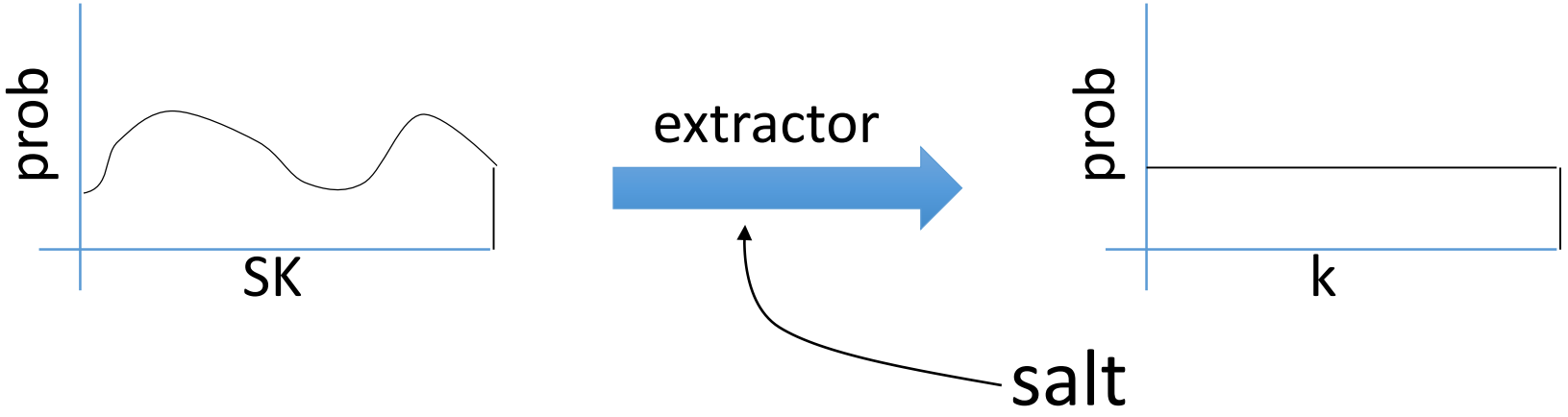
SK не равномерно распределён ⇒ PRF может и не давать случайно выглядящий выход

Примеры неравномерного распределения:

- Протоколы обмена ключей: ключ может быть равномерно распределён только в некотором подмножестве ключей
- Аппаратный PRG: возможен смещённый выход
- Пароли: …. без комментариев

Парадигма извлечения и расширения

Step 1: **извлечь** псевдослучайный ключ k из ключа SK



salt: некоторая величина

step 2: **расширить** k используя в качестве ключа PRF (как указано ранее)

HKDF: KDF from HMAC

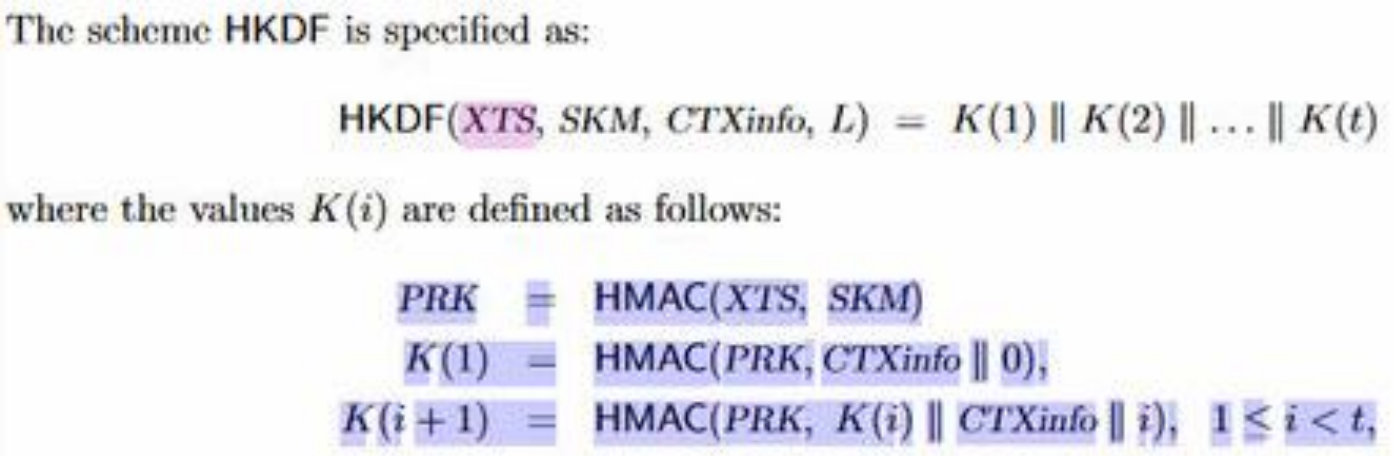
Реализует Парадигма извлечения и расширения с помощью HMAC :

- извлечение: use **k ⟵ HMAC(salt, SK)**

- Затем расширить используя HMAC в качестве PRF с ключом **k**

HKDF

Более формально



The scheme HKDF is specified as:

$$\mathsf{HKDF}(XTS, SKM, CTXinfo, L) = K(1) \parallel K(2) \parallel \ldots \parallel K(t)$$

where the values $K(i)$ are defined as follows:

$$\begin{aligned} PRK &= \mathsf{HMAC}(XTS, SKM) \\ K(1) &= \mathsf{HMAC}(PRK, CTXinfo \parallel 0), \\ K(i+1) &= \mathsf{HMAC}(PRK, K(i) \parallel CTXinfo \parallel i), \quad 1 \leq i < t, \end{aligned}$$

XTS – соль, SKM – ключевой материал, $K(i)$ – выходы

HKDF

The scheme HKDF is specified as:

$$\text{HKDF}(XTS, SKM, CTXinfo, L) = K(1) \parallel K(2) \parallel \ldots \parallel K(t)$$

where the values $K(i)$ are defined as follows:

$$\begin{aligned} PRK &= \text{HMAC}(XTS, SKM) \\ K(1) &= \text{HMAC}(PRK, CTXinfo \parallel 0), \\ K(i+1) &= \text{HMAC}(PRK, K(i) \parallel CTXinfo \parallel i), \quad 1 \leq i < t, \end{aligned}$$

XTS – случайная => PRK – стойкая PRF (т.к. HMAC – стойкая PRF), требование на хэш-функцию – необходимые требования для стойкости MAC

XTS – константа 0. PRK – детерминированная хэш-функция, на основе хэш-функции, использованной в HMAC. Для получения псевдослучайных выходов $K(i)$ необходимо использовать модель случайного оракула для хэш-функции.

Промежуточные решения – использование различных констант, счётчиков, nonce, дают промежуточные результыты.

HKDF

- Позволяет извлекать энтропию из неравномерно распределённого источника для получения равномерно распределённой последовательности
- Контекст используется для изоляции ключей между приложениями или применениями
- Соль может быть константной или отсутствовать, но случайная соль даёт лучшую стойкость. Если не получается использовать случайную – лучше использовать хотя бы счётчик
- Де-факто интернет стандарт
- Не использовать MD-5 и SHA-1

KDF для паролей (PBKDF)

- Не использовать HKDF: у паролей удивительно малая энтропия
- Полученные ключи могут быть уязвимы к перебору пол словарю исходного материала

PBKDF: **salt** и **медленное хэширование**

Пример: **PKCS#5** (PBKDF1)

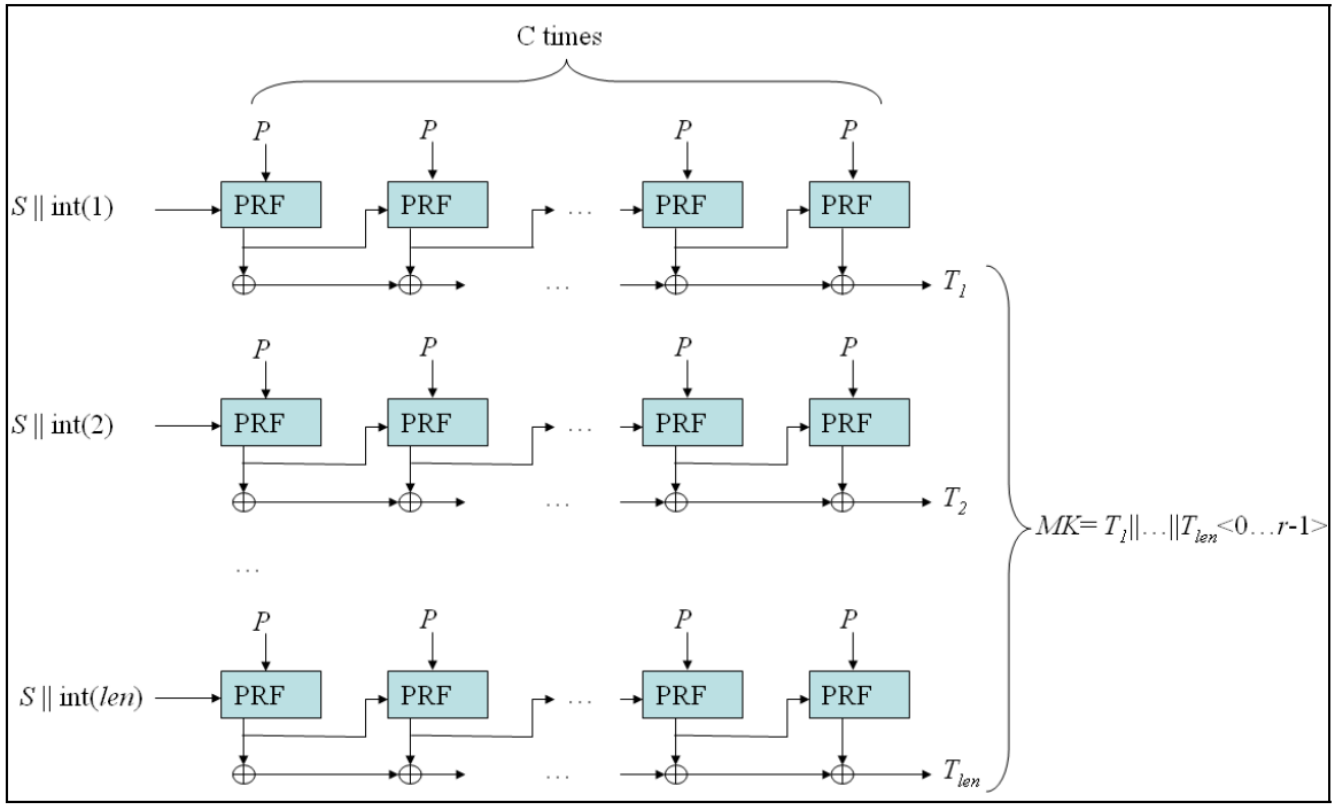
- $H^{(c)}$(pwd ll salt): вычисляем хэш-функцию с раз, подмешивая соль

Пример: PBKDF2

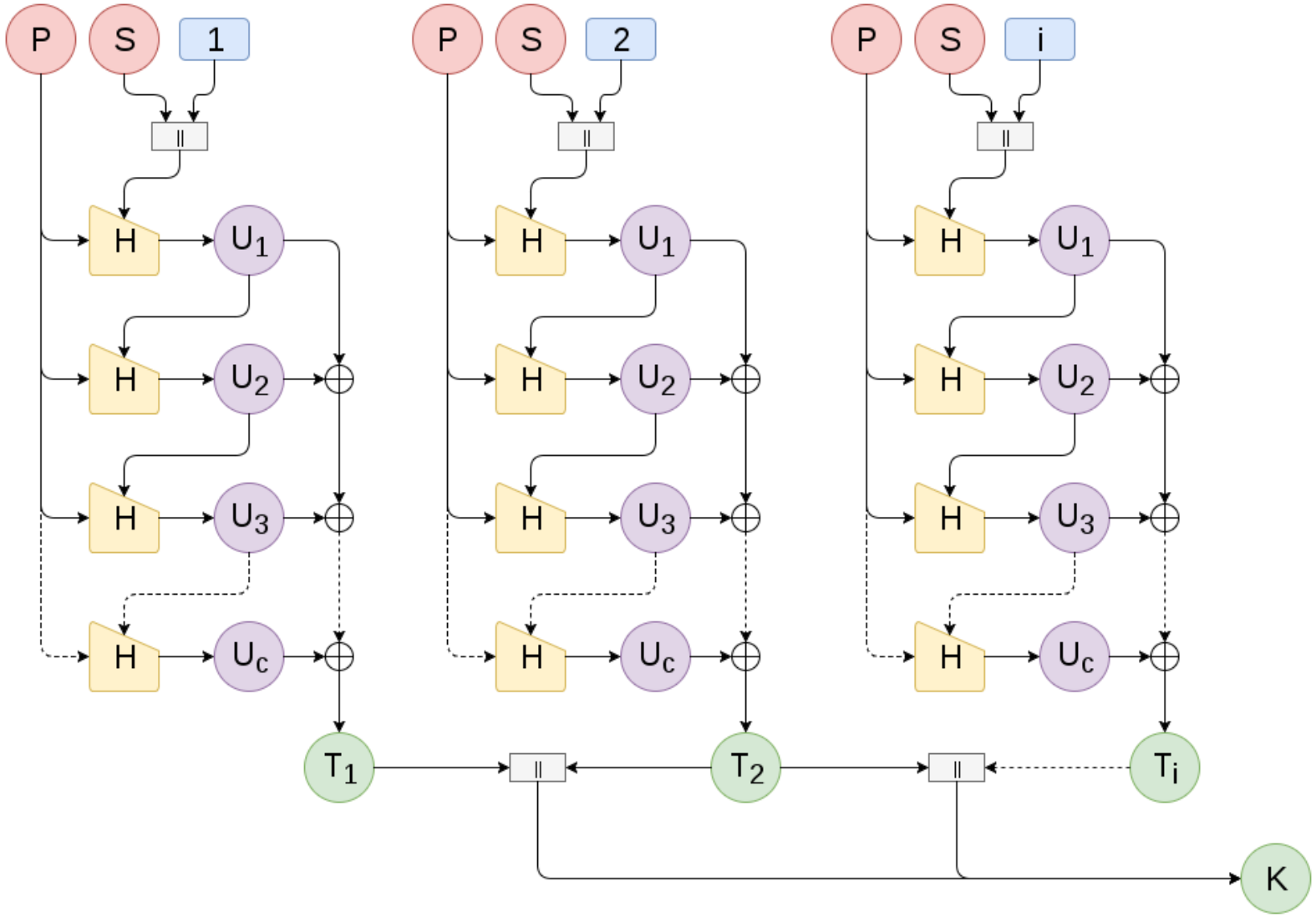
- P – пароль, S – seed, c – текущая итерация генерации ключа (при генерации нового ключа увеличивается на 1), n – число T блоков для генерации одного ключа

- $U_1 = PRF(P, S||i)$
- $U_c = PRF(U_{c-1})$
- $F(P, S, c, i) = U_1 \oplus \cdots \oplus U_c$
- $T_i = F(P, S, c, i)$
- $DK = T_1|| \ldots ||T_n$ (MK

на картинке)



PBKDF2, ещё одна картинка



PBKDF2, порядок перебора

	Master Password guessing times with hashcat's 4 GPU system		
	10000 PBKDF2 iterations (minimum for new keychains)	25000 PBKDF2 iterations (typical for new keychains)	45000 PBKDF2 iterations (high end)
	300,000 guesses/sec	120,000 guesses/sec	66,667 guesses/sec
Password Strength Entropy (in bits)			
39	9 days	23 days	41 days
52	193 years	482 years	867 years
65	1,498,426 years	3,746,064 years	6,742,915 years
78	12 billion years	29,129 billion years	52,433 billion years
90	91 trillion years	227 trillion years	408 trillion years