Хэш-функции

Макаров Артём

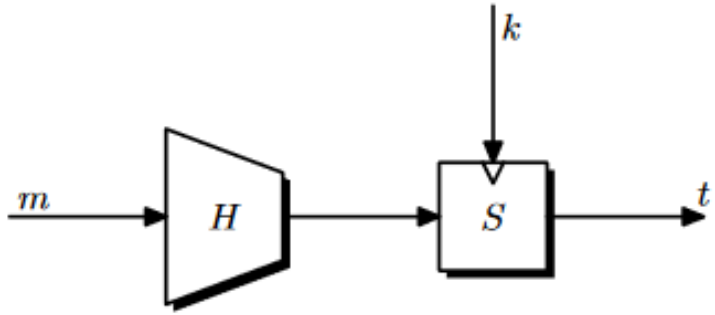
МИФИ 2024

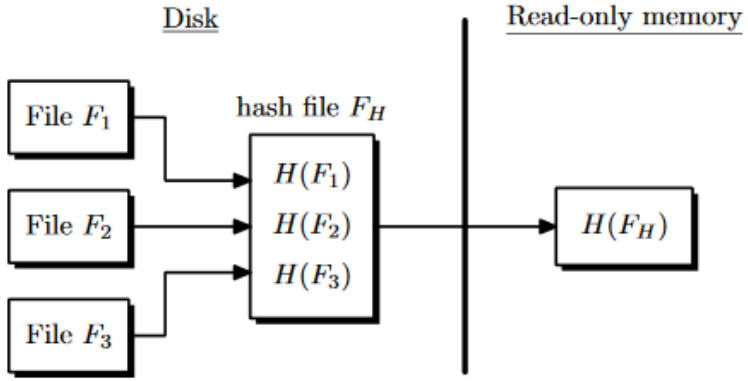
Целостность сообщений

- 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.





Атаки на основе парадокса дней рождений

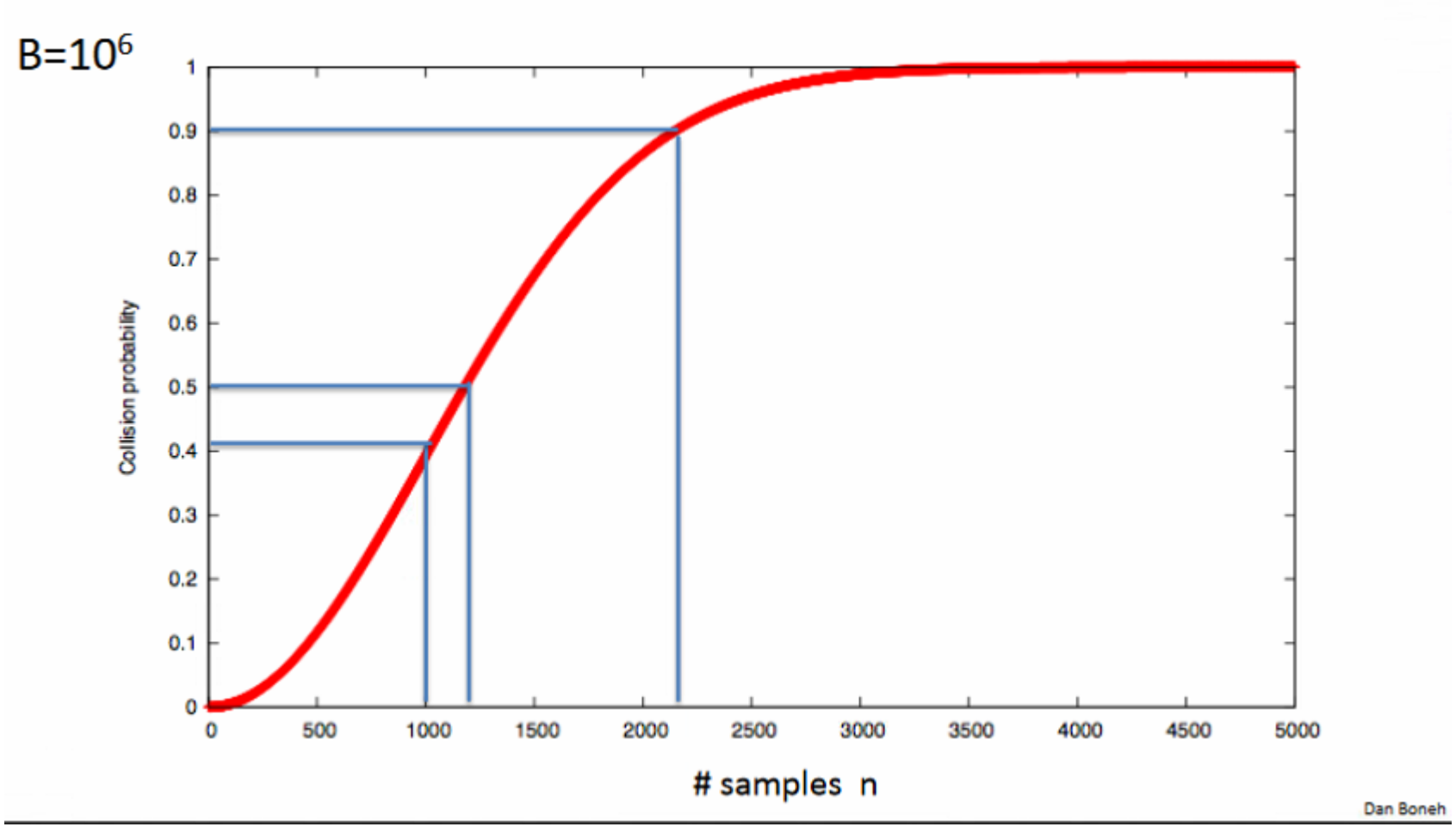
Пусть $H: M \to T$, - хэш-функция. $T = \{0,1\}^n$

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

Другие атаки на нахождение коллизий

Алгоритм	Вычислительная сложность, оп	Затраты памяти, hash_size	Реальные параметры
Угадывание коллизии	2^b	1	$M = 1, t = 2^{128}$
Использование таблиц (b-day)	$2^b / M^{1/2}$	M	$M = 2^{b/2}, t = 2^{b/2}$
Квантовый алгоритм Brassard, Høyer, Tapp (fancy ø)	$2^{b/2} / M^{1/2}$	M	$M = 1, t = 2^{b/2}$ $M = 2^{b/6}, t = 2^{5b/12}$ $M = 2^{b/3}, t = 2^{b/3}$
Параллельный алгоритм нахождения коллизий	$2^b / M^{3/2}$	$1 \times \boldsymbol{M}$	$M = 1, t = 2^b$ $M = 2^{b/6}, t = 2^{3b/4}$ $M = 2^{b/3}, t = 2^{b/2}$
Параллельный квантовый алгоритм нахождения коллизий Grover and Rudolph	$2^{b/2} / M^{1/2}$	$1 \times \boldsymbol{M}$	$M = 1, t = 2^{b/2}$ $M = 2^{b/6}, t = 2^{5b/12}$ $M = 2^{b/3}, t = 2^{b/3}$
M **параллельный** Ро-алгоритм Полларда	$2^{b/2} / M$	$1 \times \boldsymbol{M}$	$M = 1, t = 2^{b/2}$ $M = 2^{b/6}, t = 2^{b/3}$ $M = 2^{b/3}, t = 2^{b/6}$ $M = 2^{b/4}, t = 2^{b/4}$

Daniel J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?

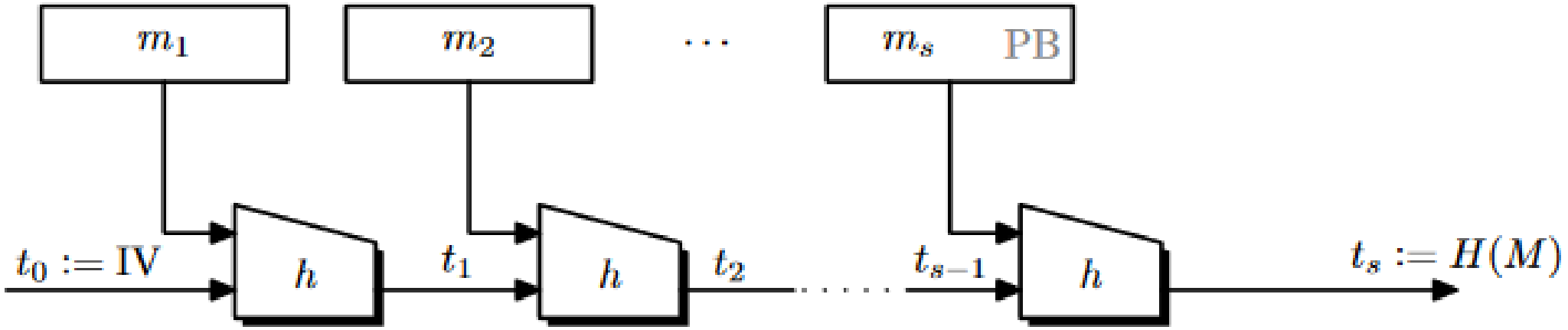
Парадигма Меркла-Дамгарда

Большинство современных хэш-функций стоится по итеративному принципу. Сначала описывается некоторая хэш-функция для сообщений малой фиксированной 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**.

SHA-1

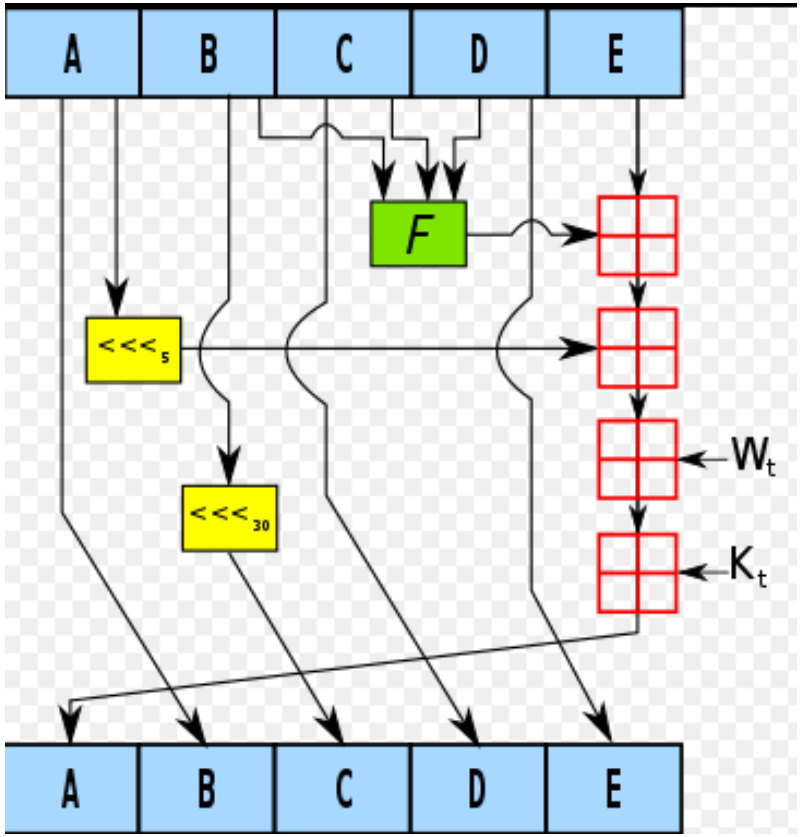
NIST 1993

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

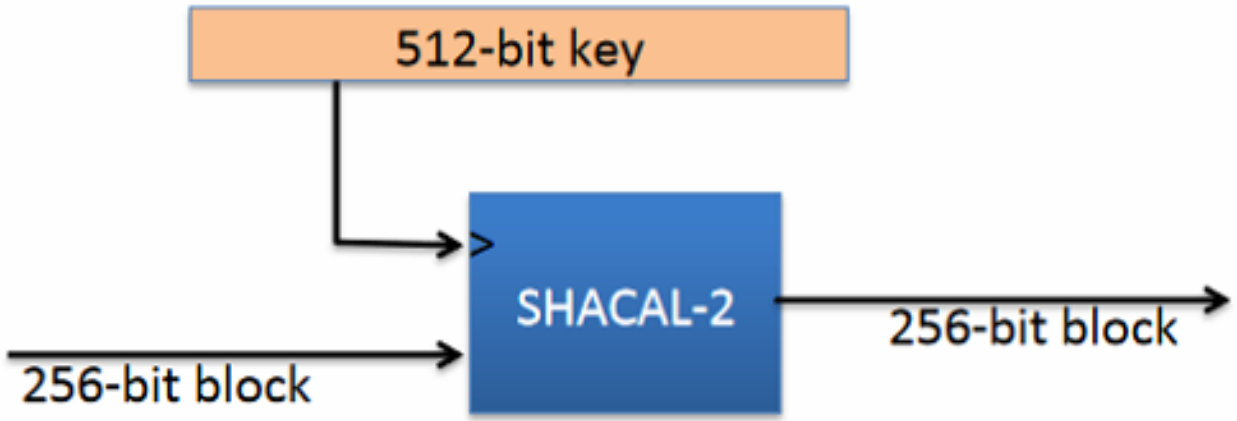
NIST 2002

Построена с использованием парадигмы Меркла-Дамгарда

Функция сжатия Девиеса-Меера

Блочный шифр – SHACAL-2

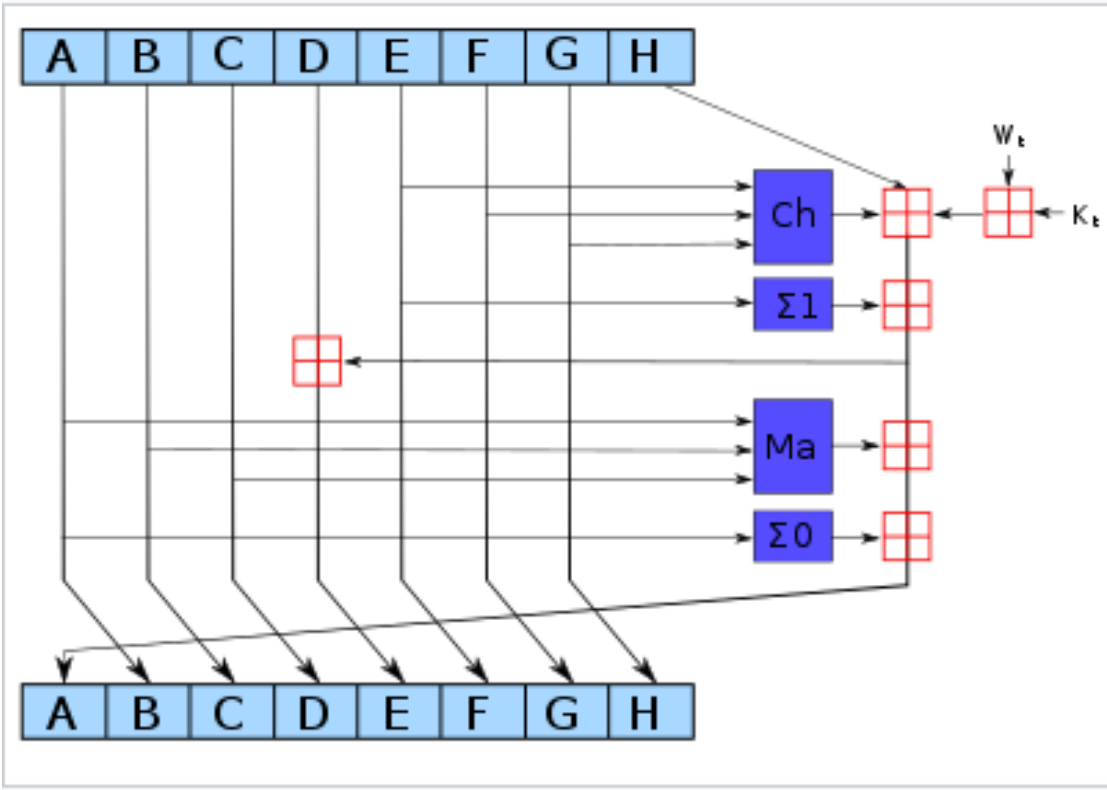
Современный стандарт хэш-функции



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

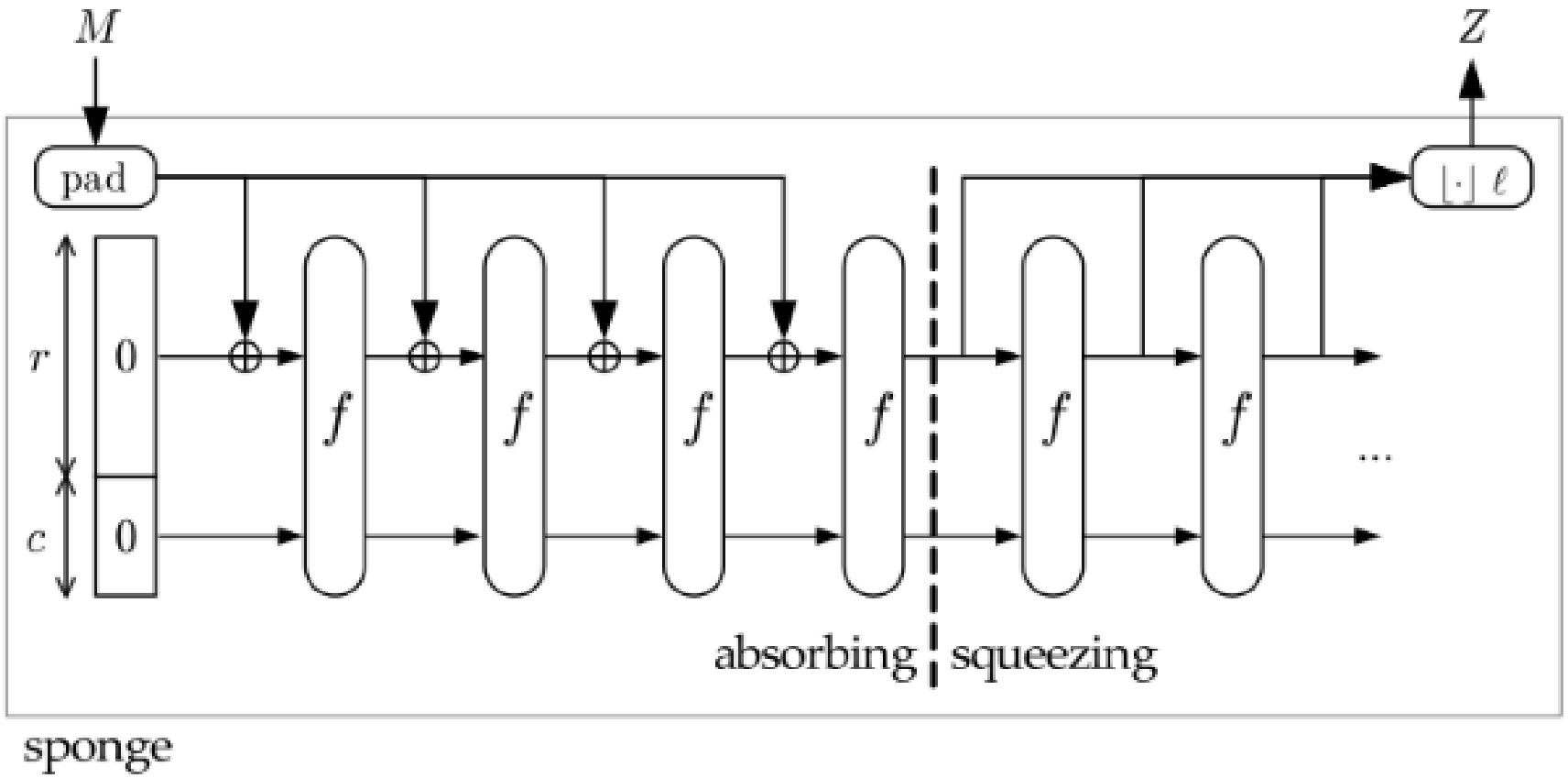
Основа губчатой конструкции – «волшебная» перестановка на некотором множестве.

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» губки

Губчатая конструкция (SHA-3)



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

Нахождение коллизий методом Полларда

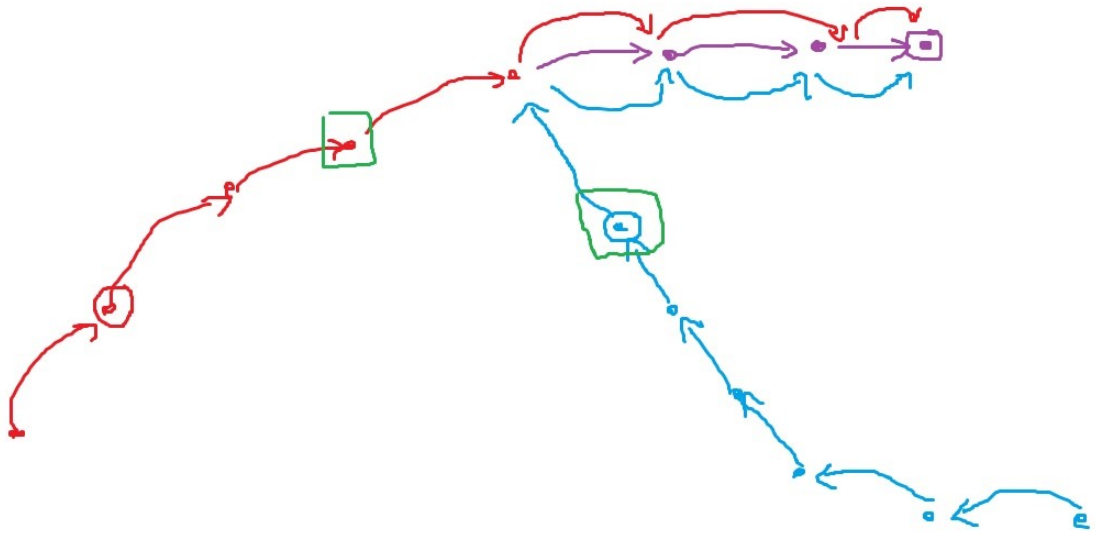
- Отличительная точка – вектор, первый q бит которого нулевые
- Имеется несколько потоков. Каждый инициализирован некоторым исходным значением y_0 (сохраняется в памяти).
- Каждый поток вычисляет последовательно цепочку хэшей с инъективной функцией $y_i = P(H(y_{i-1}))$.
- Если y_i - отличительная точка – сохраняем (i, y_i) в общую для потоков память. Если (j, y_i) уже сохранено в памяти для некоторого j => переходим ко второму этапу, иначе – продолжаем вычислять y_i

Нахождение коллизий методом Полларда

- Имеем $(i, y_i), (j, y_j)$. Пусть $i > j$, тогда $d = i - j$. Применим d итераций к начальной точке y_0 цепочки большей длины.
- Далее синхронно итерируем 2 цепочки двух потоков, пока не найдём первую коллизию (т.е. совпадение значений).
- Основная идея – если нашли коллизию на особой точке – значит коллизия была и где-то раньше.
- В качестве инъективной функции можно использовать дописывание k нулевых бит в конец вектора.

Пример нахождения коллизии на двух потоках

- 2 потока – красный и синий.
- Точка в круге – особая точка
- Нашли общую особую точку (фиолетовая в круге) за j=7, i=9.
- D= i-j=2. Итерируем длинную (синюю) цепочку 2 раза.
- Итерируем синхронно обе цепочки, пока не найдём коллизии, которые дадут точки в зелёных квадратах.



Пример нахождения коллизии на одном потоке

- Возможна ситуацию, когда поток найдёт коллизию сам на себе.
- Работаем на втором этапе, как будто с двумя цепочками, но по факту с одной.
- $(i, y_i), (j, y_i)$ – коллизия (красная точка), $d = i - j$ Итерируем цепочку d раз, оказываемся на цикле. Далее идём по циклу, ожидая коллизию