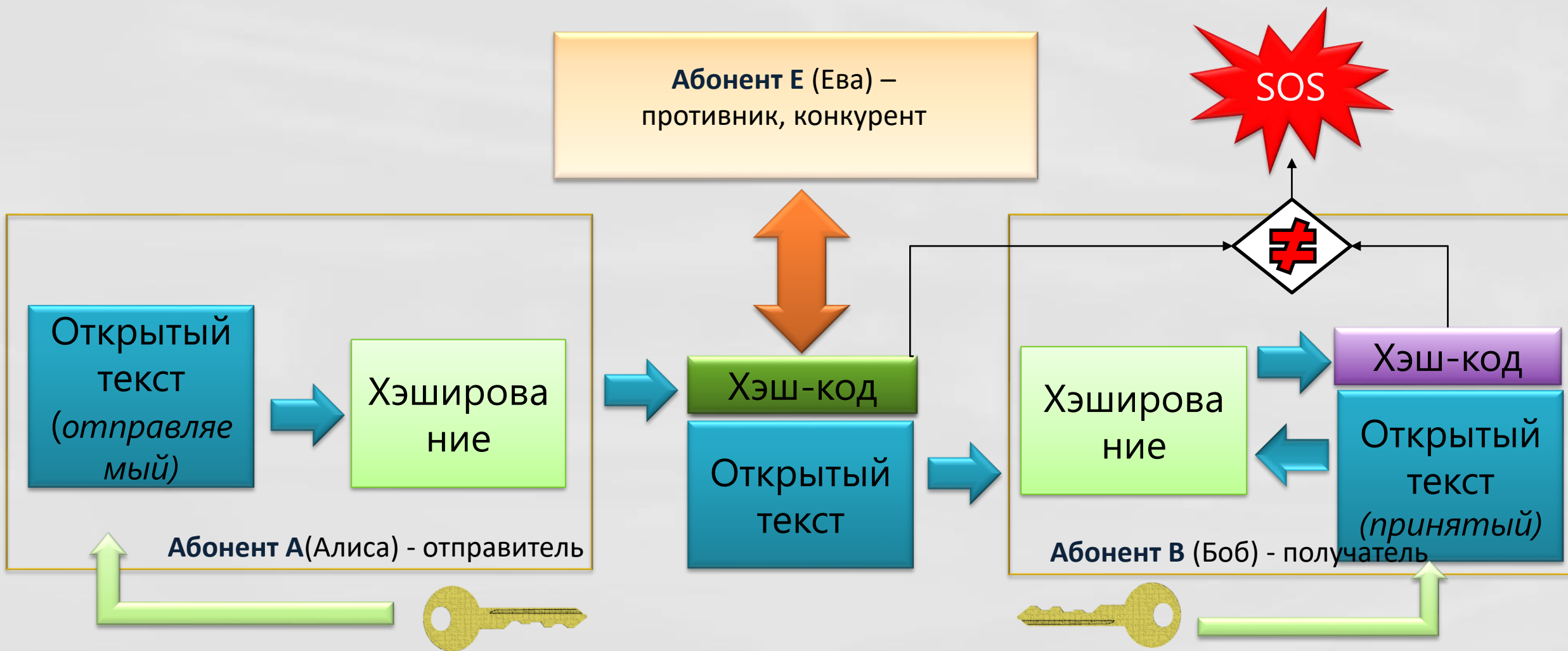


Методы контроля целостности и подлинности данных

Угрозы в фокусе темы



Базовая модель



Определения

- Хэш-функцией (*hash function*) называется математическая или иная функция, которая для строки бит произвольной длины вычисляет некоторую другую строку бит фиксированной длины (хэш-код)
- Хэш может также называться применительно к сообщению:
 - хэш
 - хэш-образ
 - свертка
 - дайджест сообщения
 - цифровой отпечаток
 - криптографическая контрольная сумма

Односторонняя хэш-функция

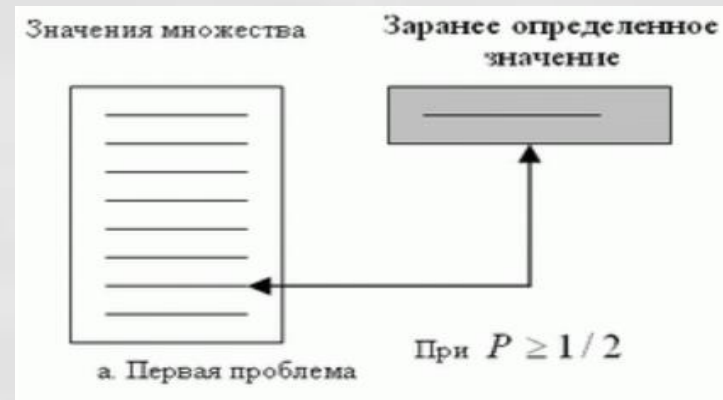
- $H(x)$ относительно легко (за полиномиальное время) вычисляется для любого значения x
- Устойчива к прообразу: для любого данного значения хэш-кода h вычислительно сложно найти x такое, что $x = H^{-1}(h)$
- Устойчива ко второму прообразу: для любого данного x вычислительно сложно найти $y: y \neq x \ \& \ H(y) = H(x)$
- Устойчива к коллизиям: вычислительно сложно найти произвольную пару (x, y) такую, что $H(y) = H(x)$ (признак сильной хэш-функции)

Понятие идеальной хэш-функции (random oracle model)

- Для нового сообщения произвольной длины вырабатывается на выходе дайджест фиксированной длины, который представляет собой случайную строку нулей и единиц. Эта пара (сообщения, дайджест) образует запись в таблице
- Когда поступает сообщение, для которого уже существует дайджест, выдается дайджест из соответствующей записи
- Дайджест для нового сообщения должен быть выбран независимо от всех предыдущих дайджестов. Это подразумевает, что модель не может использовать формулу или алгоритм для вычисления дайджеста
- Отношения между возможными сообщениями и возможными дайджестами – «многие к одному» (функциональность), поскольку длина дайджеста меньше длины сообщения
- **Пример:** Моделью идеальной хэш-функции может служить случайное назначение номера дня в году при рождении каждой персоны

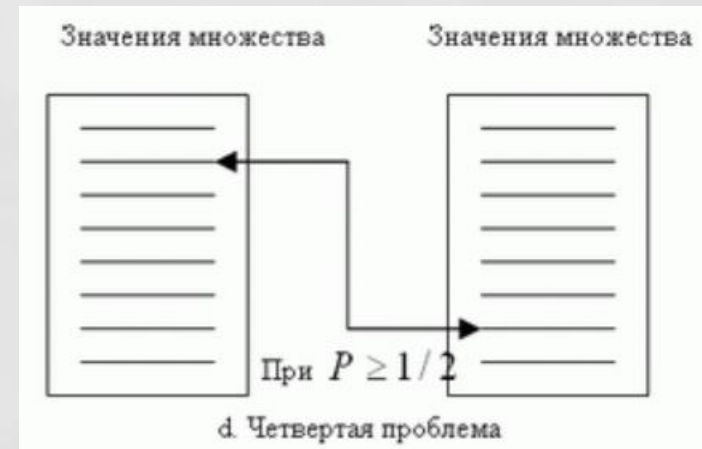
Модели атак на идеальную хэш-функцию на основе парадоксов дней рождения

- **Парадокс 1:** Каково минимальное число k - студентов в аудитории, такое, что с некоторой вероятностью по крайней мере один студент имеет заранее заданный день рождения?
- **Парадокс 2:** Каково минимальное число k - студентов в аудитории, такое, что с некоторой вероятностью по крайней мере один студент имеет тот же самый день рождения, как и студент, выбранный преподавателем?



Модели атак на идеальную хэш-функцию на основе парадоксов дней рождения

- **Парадокс 3.** Каково минимальное число k - студентов в аудитории, такое, что с заданной вероятностью по крайней мере два студента имеют тот же самый день рождения?
- **Парадокс 4.** Имеются две аудитории, каждая с k студентами. Каково минимальное значение k , такое, чтобы по крайней мере один студент из первой аудитории с некоторой вероятностью имел тот же самый день рождения, что и студент из второй аудитории?



Результаты решений проблем дней рождения

Проблема	Вероятность	Общее значение для k	Значение k при $P = 1/2$	Число студентов (при $N=365$)
1	$P = 1 - e^{-k/N}$	$k = \ln[1/(1-P)] \times N$	$k = 0,69 \times N$	253
2	$P = 1 - e^{-(k-1)/N}$	$k = \ln[1/(1-P)] \times N + 1$	$k = 0,69 \times N + 1$	254
3	$P = 1 - e^{k(k-1)/2N}$	$k = \{2 \ln [1/1-P]\}^{1/2} \times N^{1/2}$	$k = 1,18 \times N^{1/2}$	23
4	$P = 1 - e^{-k^2/2N}$	$k = \{\ln [1/1-P]\}^{1/2} \times N^{1/2}$	$k = 0,83 \times N^{1/2}$	16

- Выделенное значение (23) является решением классического парадокса дня рождения; если есть 23 студента в аудитории, то с вероятностью $P > 0,5$ двое из них родились в один день (игнорируя год их рождения).

Виды атак на хэш-функции

Атака прообраза

- Ева перехватила дайджест D ; она хочет найти любое сообщение M' , такое, что $D = h(M')$. Ева может создать список k сообщений и выполнить поиск
- Вероятность успеха, как в парадоксе 1, асимптотическая сложность атаки $O(2^n)$

Атака второго прообраза

- Ева перехватила сообщение M и хочет найти другое сообщение M' , такое, чтобы $h(M) = h(M')$. Ева может создать список из $k - 1$ сообщений и выполнить поиск
- Вероятность успеха как в парадоксе 2, асимптотическая сложность атаки $O(2^n)$

Атака коллизии

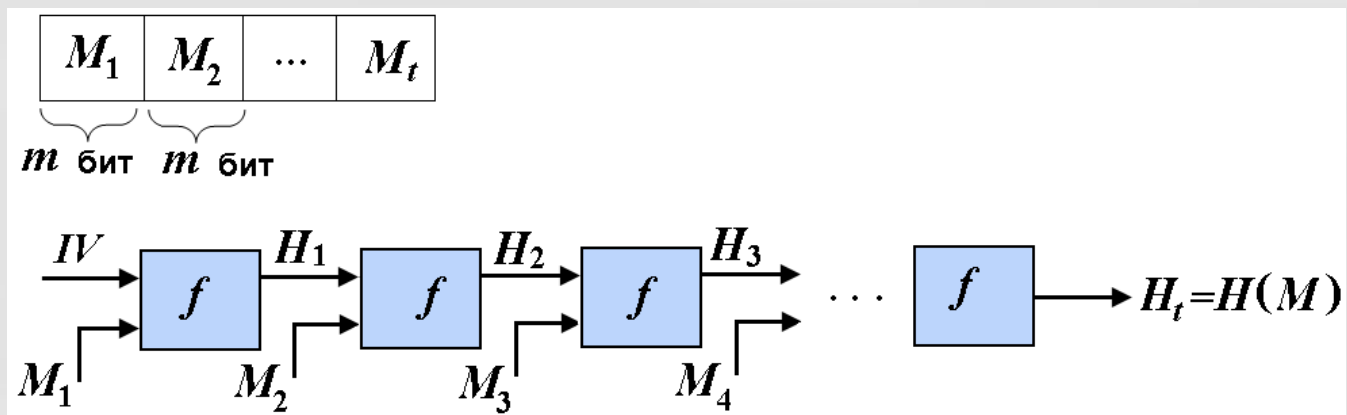
- Ева должна найти два сообщения, M и M' , такие, что $h(M) = h(M')$. Она может создать список из k сообщений и выполнить эту проверку для всех пар
- Вероятность успеха как в парадоксе 3, асимптотическая сложность атаки $O(2^{n/2})$

Дополнительная атака коллизии

- Ева создает k различных вариантов M (M_1, M_2, \dots, M_k) и k различных вариантов M' (M'_1, M'_2, \dots, M'_k). Оба набора значимы по содержанию. Затем Ева выполняет проверку для всех пар из наборов.
- Вероятность успеха как в парадоксе 4, асимптотическая сложность атаки $O(2^{n/2})$

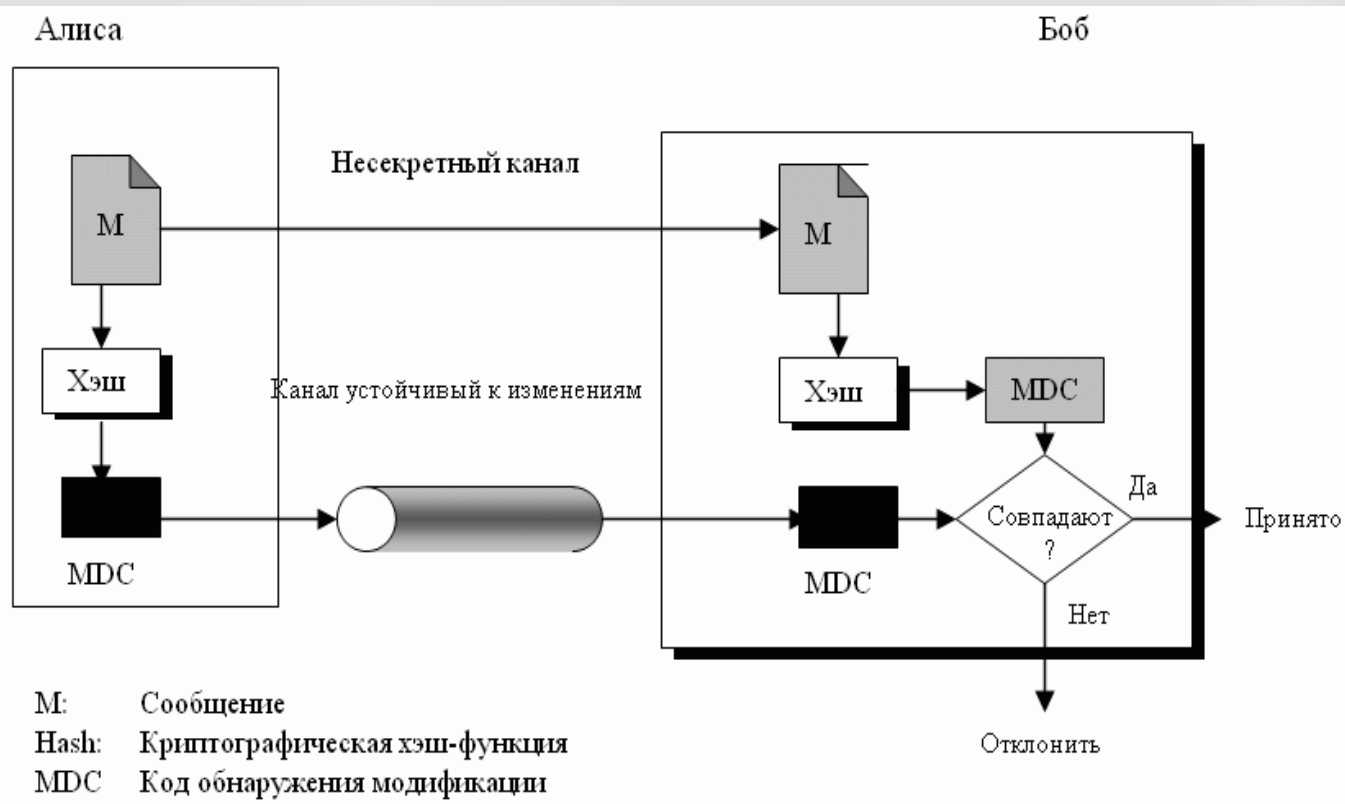
Итерированная хэш-функция

- Вычисляется по схеме Меркла – Дамгарда, основанной на многократном вычислении одношаговой функции сжатия f со входом фиксированного размера m бит



- Доказано, что если функция сжатия устойчива к коллизиям, то хэш-функция также устойчива к коллизии.

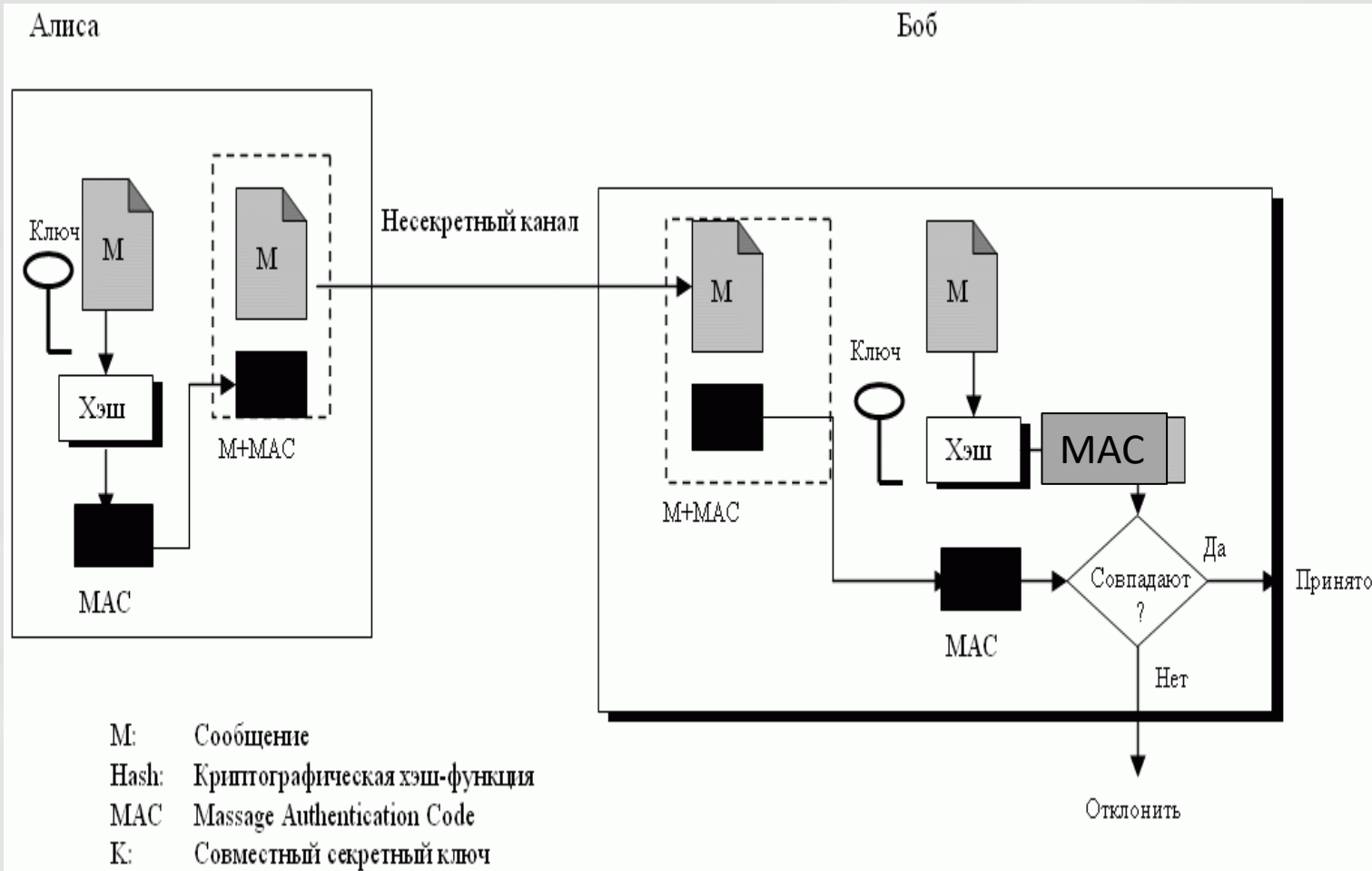
MDC- код обнаружения модификации



MDC-Modification Detection Code

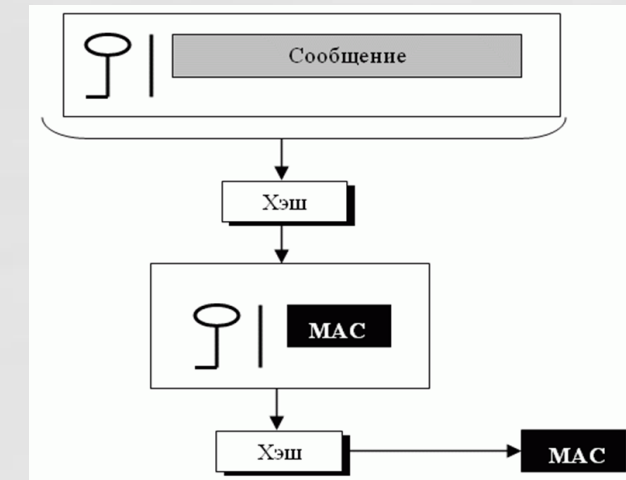
- MDC доказывает целостность сообщения, т.е. отсутствие изменений
- MDC не подтверждает подлинность отправителя сообщения
- MDC должен быть передан через безопасный (*safe*) канал не допускающий изменений

MAC - код установления подлинности сообщения



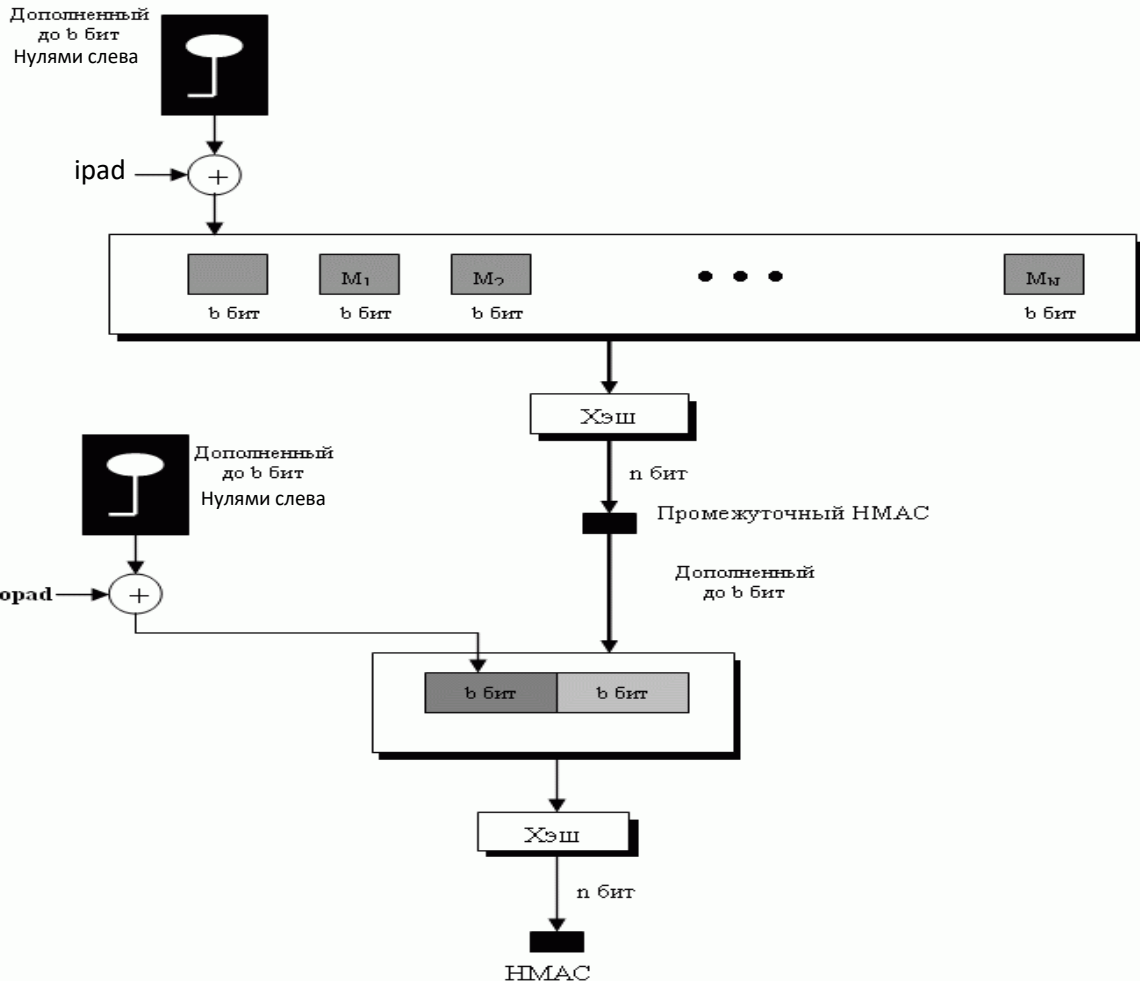
MAC-Message Authentication Code

- MAC гарантирует целостность и подлинность сообщения
- Нет необходимости использовать два канала
- Безопасность MAC зависит от безопасности основного хэш-алгоритма
- Вложенный MAC:



Nested MAC

НМАС - код, основанный на хэшировании



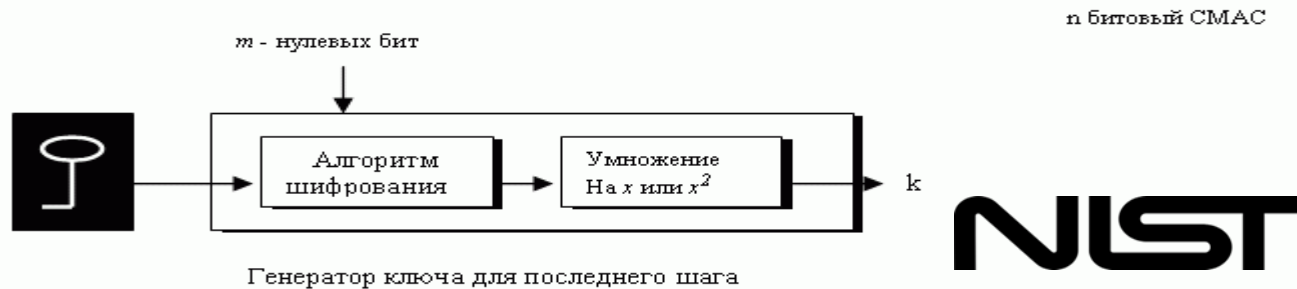
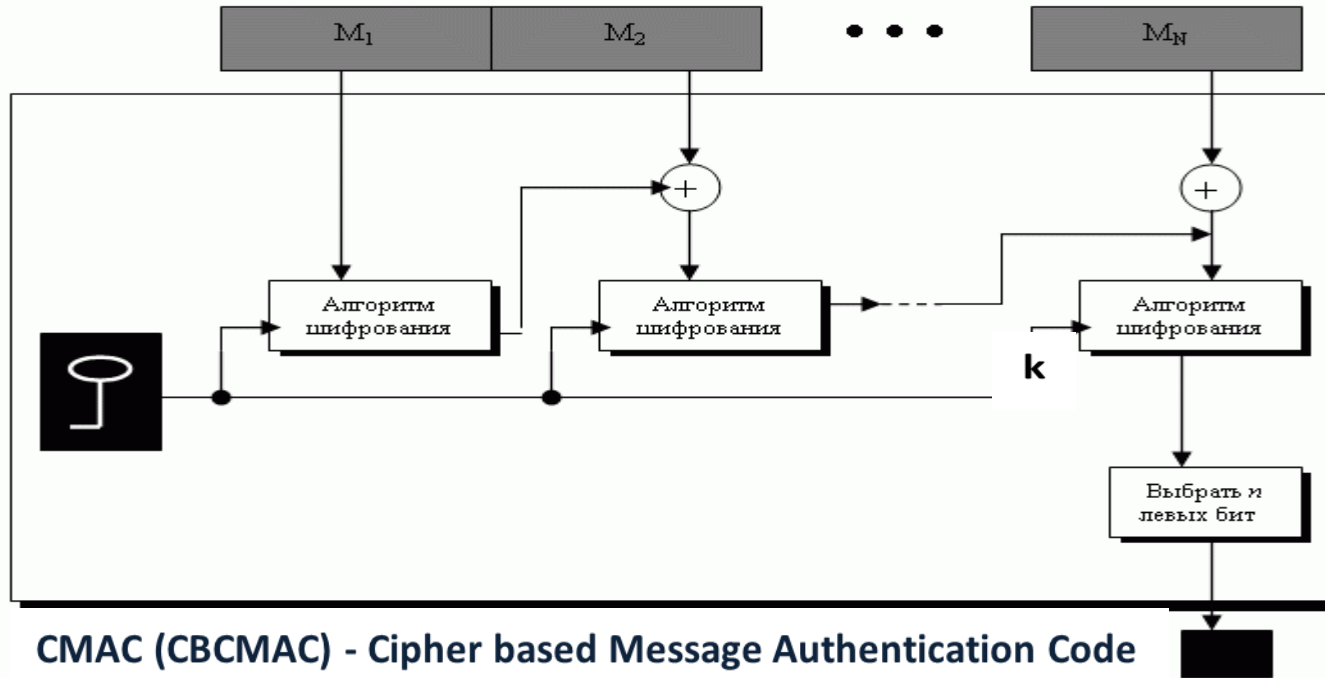
- *ipad* – константа длиной b бит из повторяющихся байт 00110110
- *opad* – константа длиной b бит из повторяющихся байт 01011100

HMAC - Hash-based Message Authentication Code

NIST

СМАС - код на основе шифрования базового сообщения

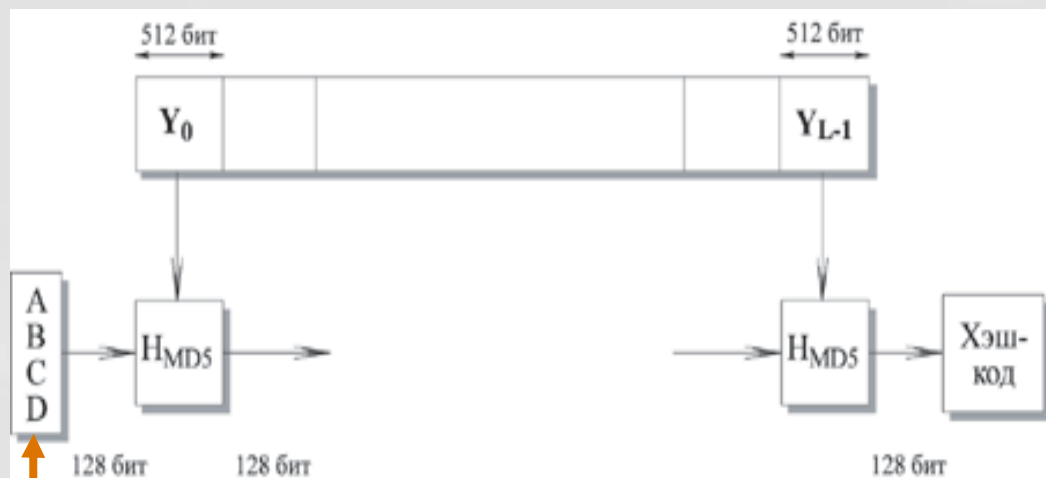
M_i : Блок сообщения i



- Цель - создать один блок СМАС из N блоков исходного текста, используя N раз шифрование с симметричным ключом (режим *CBC - Cipher Block Chaining*)
- Неполный последний блок дополняется единичным битом (1) и достаточным количеством нулевых бит (0), чтобы сделать его m -битовым
- Ключ последнего шага k – результат шифрации и умножения на x , если нет никакого дополнения блока M_N , или на x^2 , если дополнение есть. Умножение проводится в $GF(2^m)$ с неприводимым полиномом степени m , выбранным в соответствии с используемым конкретным протоколом

NIST

Хэш-функция MD5

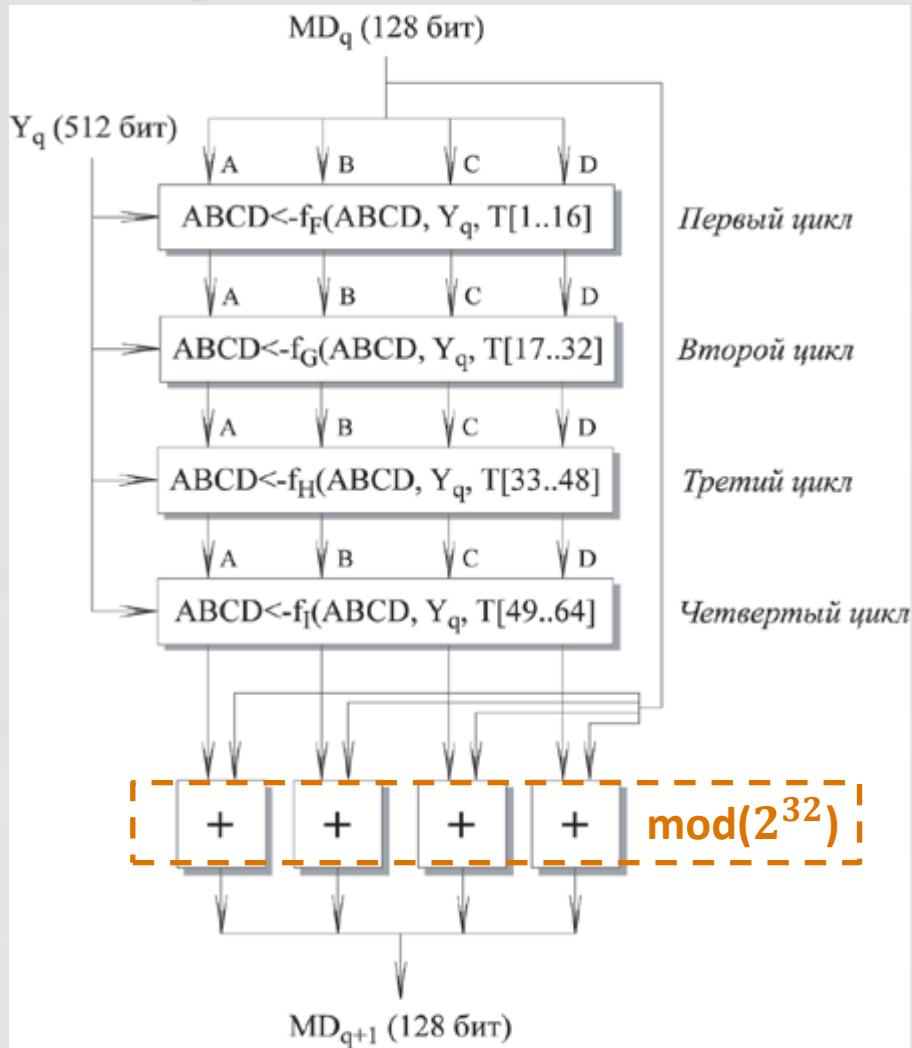


Сообщение	Добавление от 1 до 448 бит	Длина исходного сообщения L_0 - 64 бит
-----------	-------------------------------	--

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

- Добавление в последний блок недостающих битов 100000....0
- Добавление в последний блок 64 битного представления длины исходного сообщения L_0
$$L_0 = L_0 \bmod (2^{64})$$
- В итоге длина сообщения кратна 512
- Инициализация MD буфера (4 регистра по 32 бита)

Функция сжатия H_{MD5}



- Каждый цикл переопределяет значение буфера ABCD
- T - массив вычисляемых величин (по 32 бита):

$$T_i = \text{int} (2^{32} * \text{abs}(\sin(i))), i=1,64$$

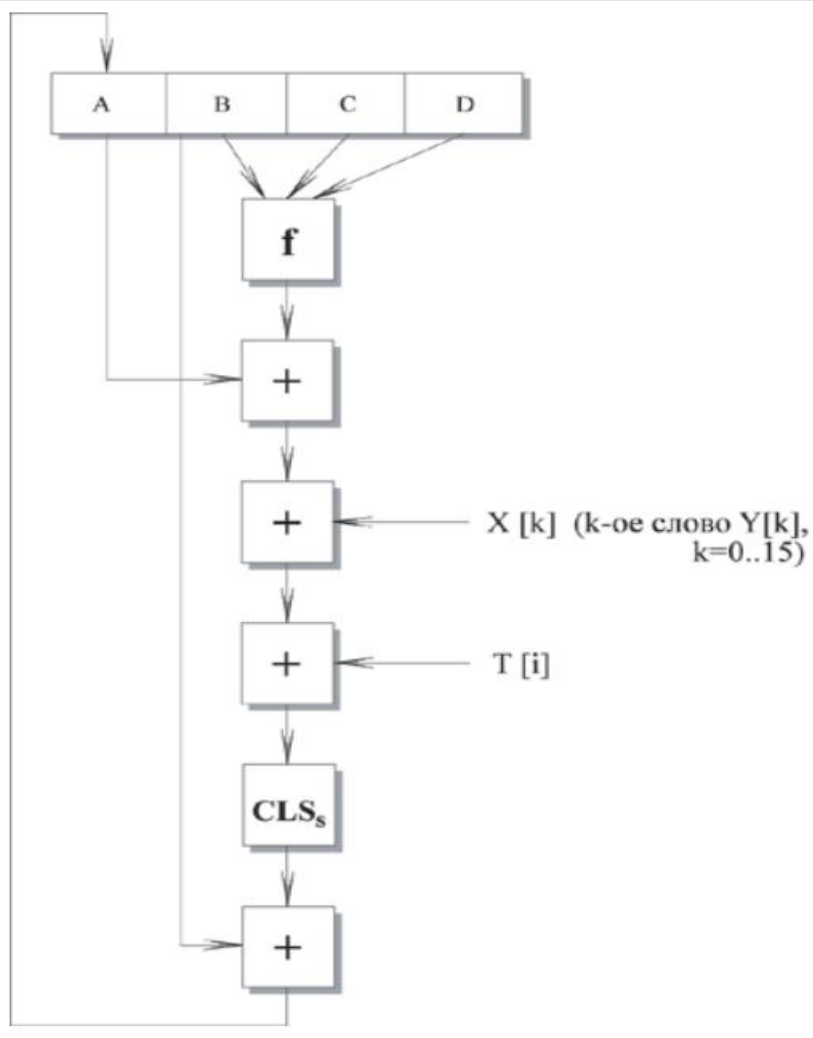
где :

$\text{int} ()$ –целая часть числа

$\text{abs}()$ – абсолютное значение

$\sin()$ - синус

Цикл сжатия H_{MD5}



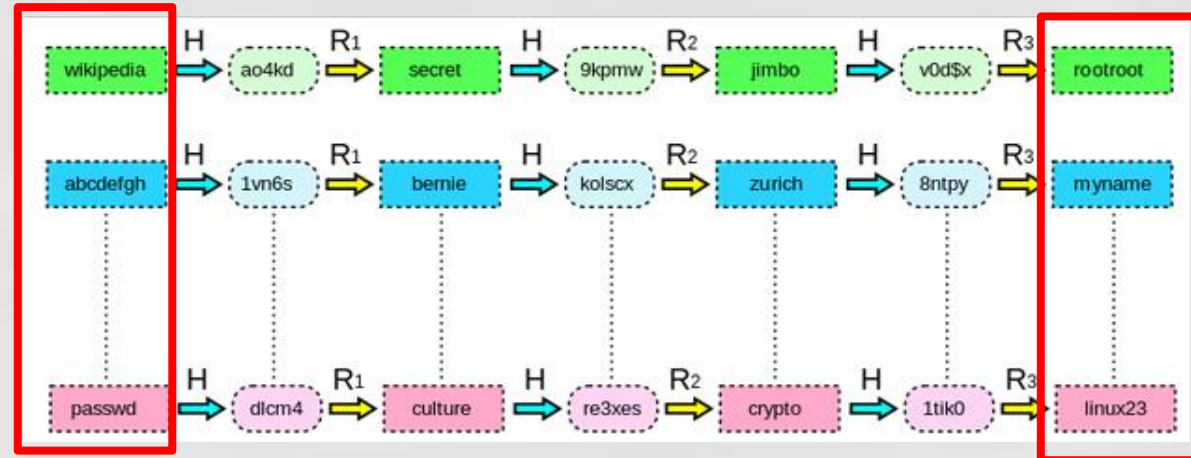
- Каждый цикл состоит из 6 шагов
- f – одна из элементарных функций
 - $f_F = (B \& C) \vee (\text{not } B \& D)$
 - $f_G = (B \& D) \vee (C \& \text{not } D)$
 - $f_H = B \oplus C \oplus D$
 - $f_I = C \oplus (B \& \text{not } D)$
- Сложение выполняется по модулю 2^{32}
- CLS_5 циклический сдвиг влево на 5 разрядов

Свойства MD5

- Каждый бит хэш-кода является функцией от каждого бита входа
- Комплексное повторение элементарных функций обеспечивает хорошее перемешивание результата
- MD5 является наиболее сильной хэш-функцией для 128-битного хэш-кода

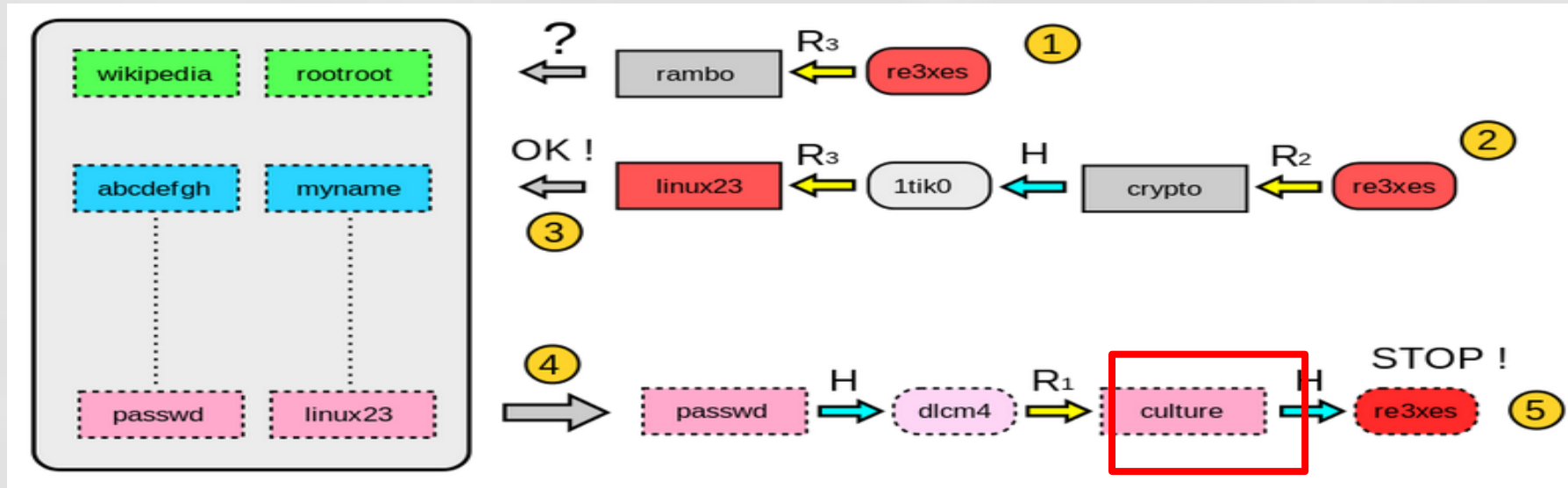
Пример атаки на MD5

- Атака подбора пароля к известному хэш-коду h с использованием «Радужных таблиц» (RainbowTables)
- Дополнительные исходные данные:
 - Алфавит пароля A
 - Длина пароля n
 - Используемая хэш-функция H (например, MD5)
 - Набор функций редукции $R_k: (H(x), k) \rightarrow A^n$
- Этап 1: создаем «радужную» таблицу возможных паролей (пример для $k=3$)
 - Запоминаем только крайние столбцы
 - Убираем строки с коллизиями
 - Добавляем новые строки



Пример атаки на MD5

- Этап 2: подбираем строку с хэш-кодом равным заданному ($h = \text{re3xes}$):



- Вычисляем $R_k(h)$, $R_k(H(R_{k-1}(h)))$, пока не найдем совпадение в последнем столбце (шаги 1, 2, 3)
- Восстанавливаем цепочку и ищем хэш-код со значением h
- Предшествующее h значение в цепочке есть искомый пароль (шаг 5)

Хэш-функция SHA-1



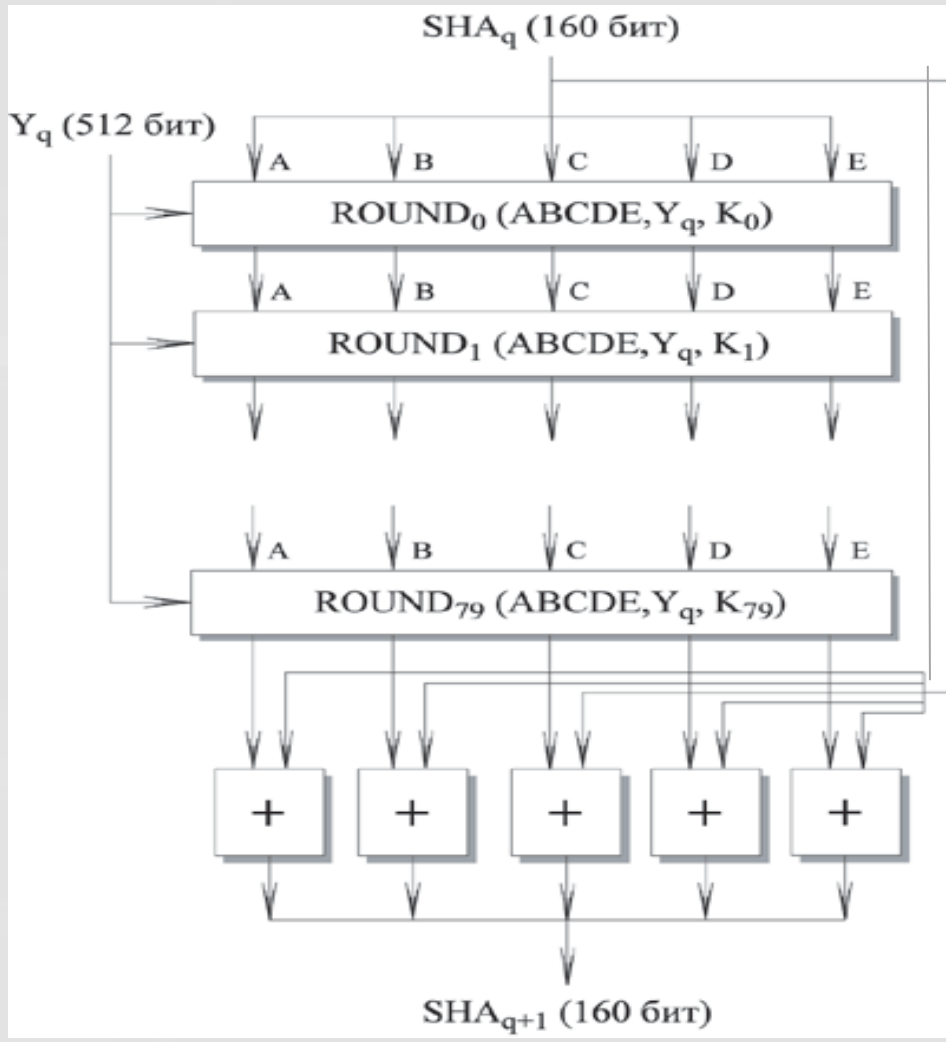
L_0 - 64 бит

A = 67452301
B = EFCDA89
C = 98BADCFE
D = 10325476
E = C3D2E1F0

NIST

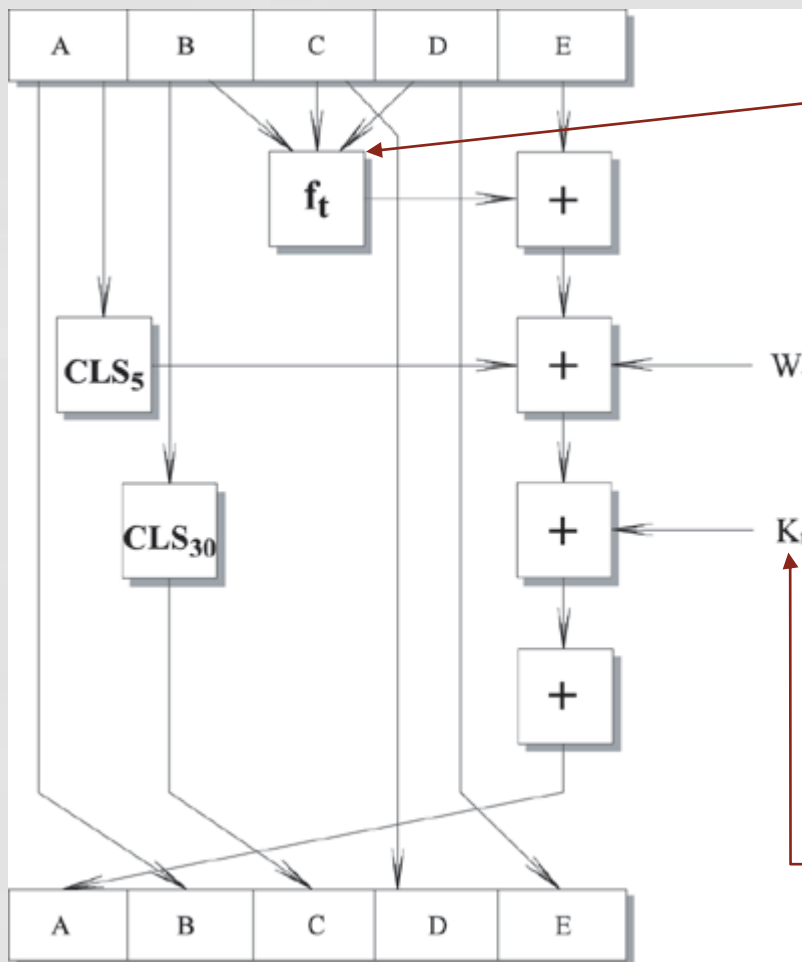
- Добавление недостающих битов
- Добавление 1000000....0
- Добавление длины исходного сообщения L_0
- Инициализация MD буфера

Функция сжатия H_{SHA-1}

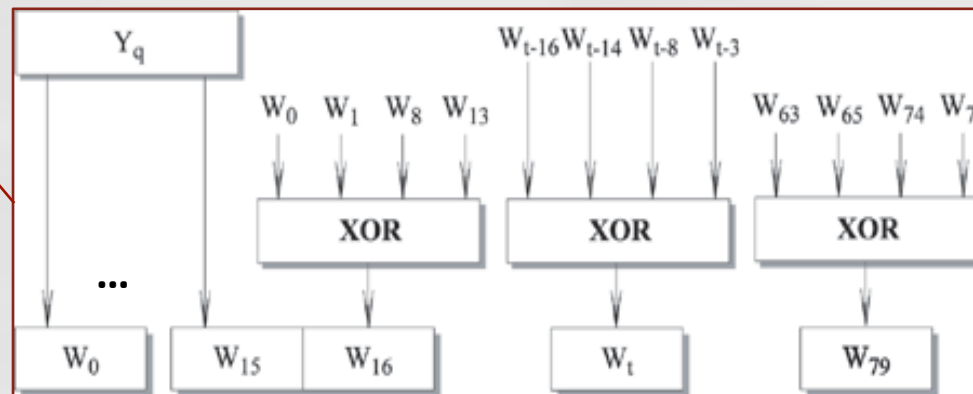


- Состоит из 80 циклов обработок
- Все 80 циклов обработок имеют одинаковую структуру
- Каждый цикл переопределяет 160-битное значение буфера ABCDE
- В каждом цикле используется дополнительная константа K_t принимающая 4 различных значения
- Сложение выполняется по модулю 2^{32}

Цикл сжатия H_{SHA-1}



Номер цикла	$f_t(B, C, D)$
$(0 \leq t \leq 19)$	$(B \wedge C) \vee (\neg B \wedge D)$
$(20 \leq t \leq 39)$	$B \oplus C \oplus D$
$(40 \leq t \leq 59)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$(60 \leq t \leq 79)$	$B \oplus C \oplus D$



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

$0 \leq t \leq 19 \quad K_t = 5A827999$
 $20 \leq t \leq 39 \quad K_t = 6ED9EBA1$
 $40 \leq t \leq 59 \quad K_t = 8F1BBCDC$
 $60 \leq t \leq 79 \quad K_t = CA62C1D6$

Сравнение SHA-1 и MD5

	<i>MD5</i>	<i>SHA-1</i>
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 <i>логических функций</i>	4	3
Число дополнительных констант	64	4

- Если предположить, что оба алгоритма не содержат каких-либо структурированных данных, которые уязвимы для криптоаналитических атак, то SHA-1 является более стойким алгоритмом
- SHA-1 содержит больше шагов (80 вместо 64) и выполняется на 160-битном буфере по сравнению со 128-битным буфером MD5. Таким образом, SHA-1 должен выполняться приблизительно на 25% медленнее, чем MD5 на той же аппаратуре 32-битной архитектурой

Хэш-функции SHA-2

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

Хэш-функция ГОСТ Р 34.11-12

- ГОСТ Р 34.11-2012 — действующий российский криптографический стандарт, определяющий алгоритм и процедуру вычисления хэш-функции
- Разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС»
- Этот стандарт разработан и введён в качестве замены устаревшему стандарту ГОСТ Р 34.11-94
- Проектное название Стрибог-256, Стрибог-512 для хэш значений 256 и 512 бит соответственно

Общие характеристики хэш-функции

Размер входа: 512 бит

Размер выхода: 512 и 256 бит соответственно

Число раундов: 12

Параметры настройки:

$IV = 0^{512}$ – для функции с длиной хэш-кода 512 бит

$IV = (00000001)^{64}$ – для функции с длиной хэш-кода 256 бит

$h := IV$

$N := 0^{512}$ счетчик бит

$\Sigma := 0^{512}$ контрольная сумма блоков

Основные этапы

Инициализация

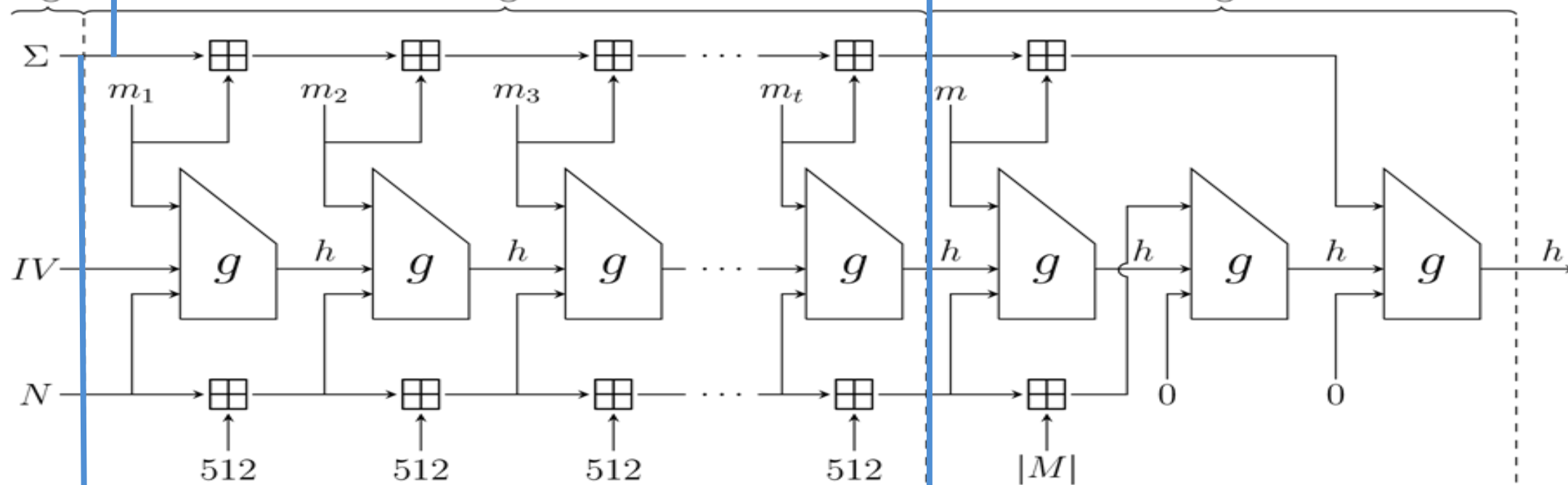
Stage 1

Сжатие

Stage 2

Завершение

Stage 3



$$\Sigma = N = 0^{512}$$

$$IV = \begin{cases} 0, & \text{Stribog-512} \\ (00000001)^{64}, & \text{Stribog-256} \end{cases}$$

$$t = \left\lfloor \frac{|M|}{512} \right\rfloor$$

$$m = 0^{512-|M|} || 1 || M$$

$$H = \begin{cases} h, & \text{Stribog-512} \\ MSB_{256}(h), & \text{Stribog-256} \end{cases}$$

Обозначения

$0^{512} = (000 \dots 0)$ – конкатенация 512 экземпляров 0

$1 || m$ – конкатенация 1 и блока сообщения

$:=$ – операция присвоения

\oplus – операция покомпонентного сложения по модулю 2 двух двоичных векторов одинаковой размерности

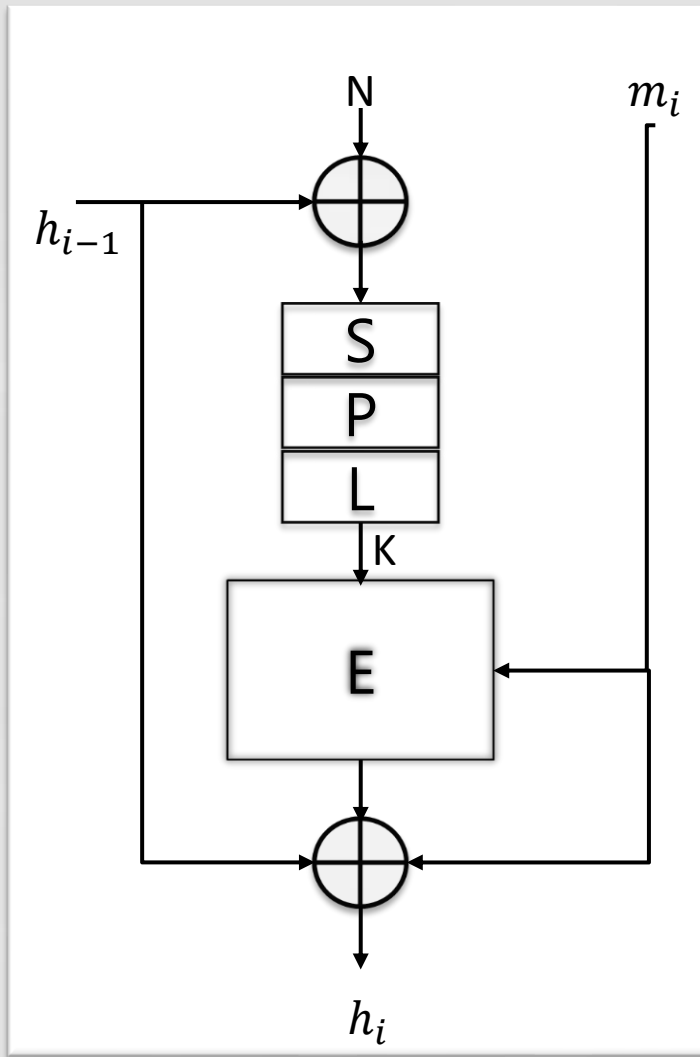
\boxplus – операция сложения в кольце вычетов по модулю 2^{512}

IV – вектор инициализации

$C_1 \dots C_{12}$ – итерационные константы

MSB_n – старшие n бит

Функция сжатия g



- S (***Substitution***) - нелинейная биекция: аргумент рассматриваются как массив из 64 байт, заменяемых по заданной стандартной таблице подстановки
- P (***Permutation***) - переупорядочивание байт по определённому в стандарте порядку
- L (***Linear transformation***) - линейное преобразование: аргумент рассматривается как 8x64-битных векторов, которые заменяются результатом умножения на определённую стандартной матрицу 64×64 над $GF(2)$
- E (***Encryption***) – функция симметричного шифрования

• <http://habrahabr.ru/post/188152/>

Псевдокод функции шифрования $E(K, m)$

- **state** = $K \oplus m$;
- Для $i := 0$ по 11 выполнить:
 - $state := S(state)$;
 - $state := P(state)$;
 - $state := L(state)$;
 - $K := \text{KeySchedule}(K, i)$;
 - $state := state \oplus K$;
- Вернуть **state** в качестве результата

Функция *KeySchedule*(K, i)

$K := K \oplus C[i]$ // C набор 512-битных констант

$K := S(K)$;

$K := P(K)$;

$K := L(K)$;

Вернуть **K** в качестве результата

Сравнение ГОСТ Р 34.11-94 и 34.11-2012

Критерий сравнения	ГОСТ Р 34.11-94	ГОСТ Р 34.11-2012
Размер блока сообщения	256 бит	512 бит
Внутреннее состояния хэш-функции	256 бит	512 бит
Длина хэш-кода	256 бит	512 или 256 бит
Функция сжатия	Симметричный блочный шифр ГОСТ Р 28147-89	SPL-преобразования
Дополнение (падинг)	(00...0)	(00...01)
Вектор инициализации	не определен	0^{512} или $(00000001)^{64}$

Применение

- Обязателен к применению в качестве алгоритма хэширования в государственных и ряде коммерческих организаций;
- Реализация систем цифровой подписи
- В сертификатах открытых ключей;
- Для защиты сообщений в S/MIME (Cryptographic Message Syntax, PKCS#7);
- Для защиты соединений в TLS (SSL, HTTPS, WEB);
- Для защиты сообщений в XML Signature (XML Encryption);
- Для защиты целостности Интернет адресов и имён (DNSSEC).

Конкурс SHA-3

- Предпосылки: в 2006 году возникло предположение, что в будущем надежность хэш-функции SHA-2 значительно снизится по следующим причинам:
 - Рост мощности и производительности вычислительных устройств
 - Появления новых методов криптоанализа
- 2 ноября 2007 года NIST инициировал разработку одного или нескольких дополнительных алгоритмов хэширования (кандидатов на новый стандарт) через открытый конкурс, подобный конкурсу AES



Требования к новой хэш-функции

• Безопасность

- Соответствие общим для хэш-функций требованиям
- Устойчивость к известным атакам
- Применимость в качестве НМАС, генератора псевдослучайных чисел

• Производительность

- Скорость работы и требования к памяти для большого числа 32х и 64х-битных платформ, для платформ портативных устройств, возможность оптимизации на многоядерных архитектурах
- Скорость работы на конечных устройствах: ПК, мобильных устройствах (точки доступа, роутеры, портативные медиаплееры, мобильные телефоны и терминалы оплаты)

• Алгоритм и характеристики реализации

- Гибкость дизайна-включает в себя возможность использования на большом числе платформ, учет возможности расширения набора инструкций процессора и распараллеливания
- Простота дизайна - оценивалась по сложности анализа и понимания алгоритма, так как простой дизайн дает больше уверенности в оценке безопасности алгоритма.

Этапы конкурса

- Список из 64 кандидатов, прошедших в первый тур, был опубликован 9 декабря 2008 года
- Список из 14 кандидатов, прошедших во второй тур, был опубликован 24 июля 2009 года
- Список из 5-ти финалистов объявлен 10 декабря 2010 года: *BLAKE, Grøstl, JH, Keccak, Skein*
- Победитель был объявлен 2 октября 2012 года, им стал алгоритм *Keccak (SHA-3)*
- 5 августа 2015 года алгоритм утверждён и опубликован в качестве стандарта FIPS 202

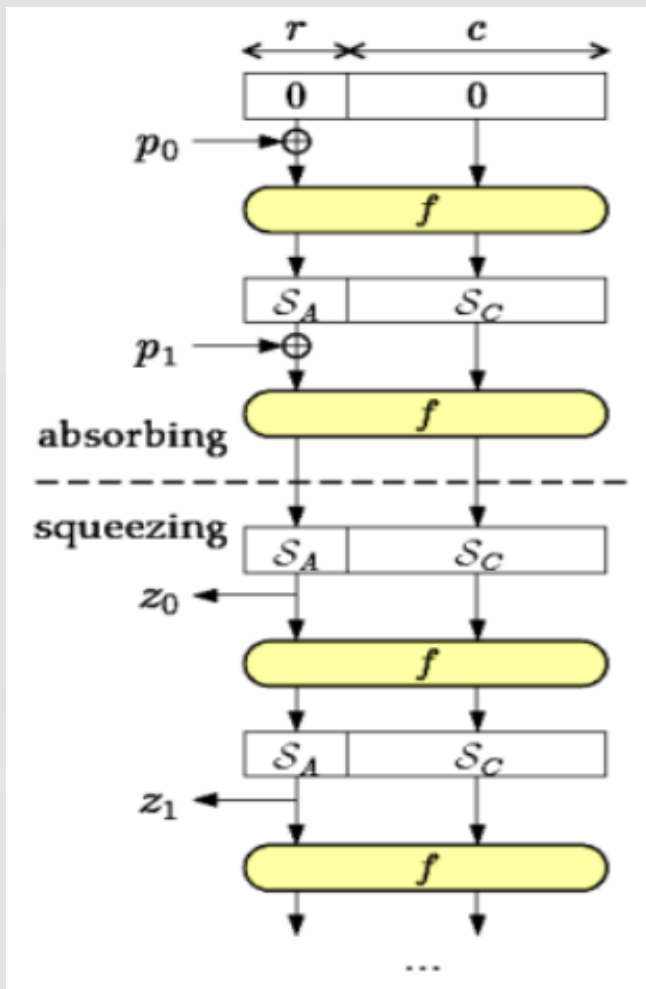
Хэш-функция Кессак

Разработка группы из четырёх криптологов компаний
STMicroelectronics и NXP Semiconductors

Преимущества победителя

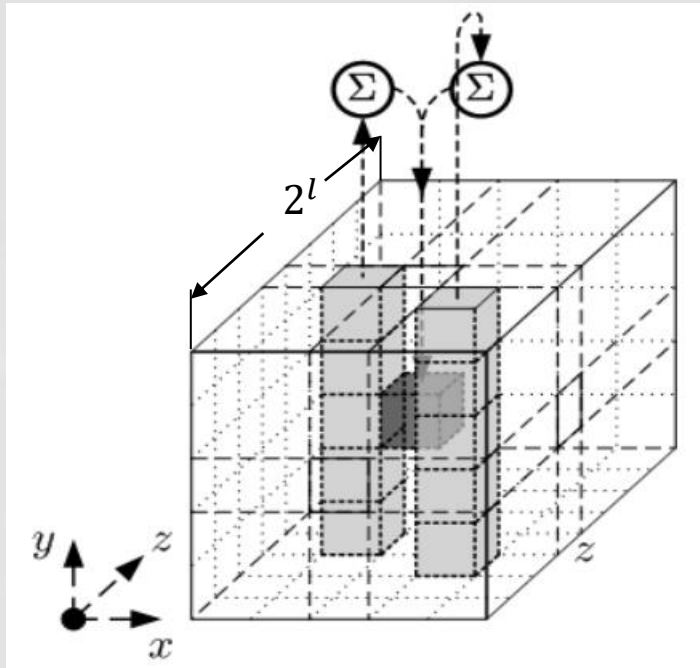
- Самый производительный в аппаратной реализации среди финалистов
- Основан на оригинальном метод шифрования - функция «губка» (поэтому, атаки, рассчитанные на SHA-2, не сработают)
- Существенным преимуществом является возможность реализации на миниатюрных встраиваемых устройствах (например, USB-флэш-накопитель).

Хэш-функция Кессак



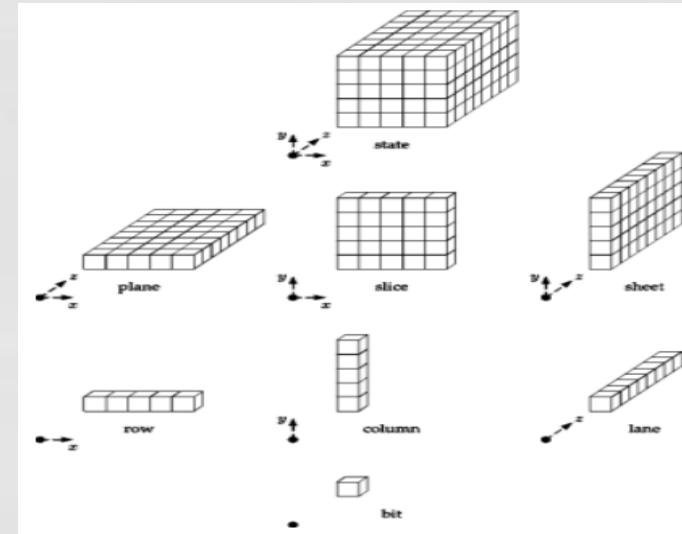
- В основе Кессак (произносится как "Ketchak") лежит конструкция под названием *Sponge* (губка)
- Алгоритм состоит из двух этапов:
 - Absorbing (впитывание). На каждом шаге очередной блок сообщения p_i длиной r подмешивается к части внутреннего состояния S , которая затем целиком модифицируется функцией f - многораундовой бесключевой псевдослучайной перестановкой
 - Squeezing (отжатие). Чтобы получить хэш, функция f многократно применяется к состоянию, и на каждом шаге извлекается и сохраняется кусок размера r до тех пор, пока не получим выход Z необходимой длины (путем конкатенации).

Внутреннее состояние



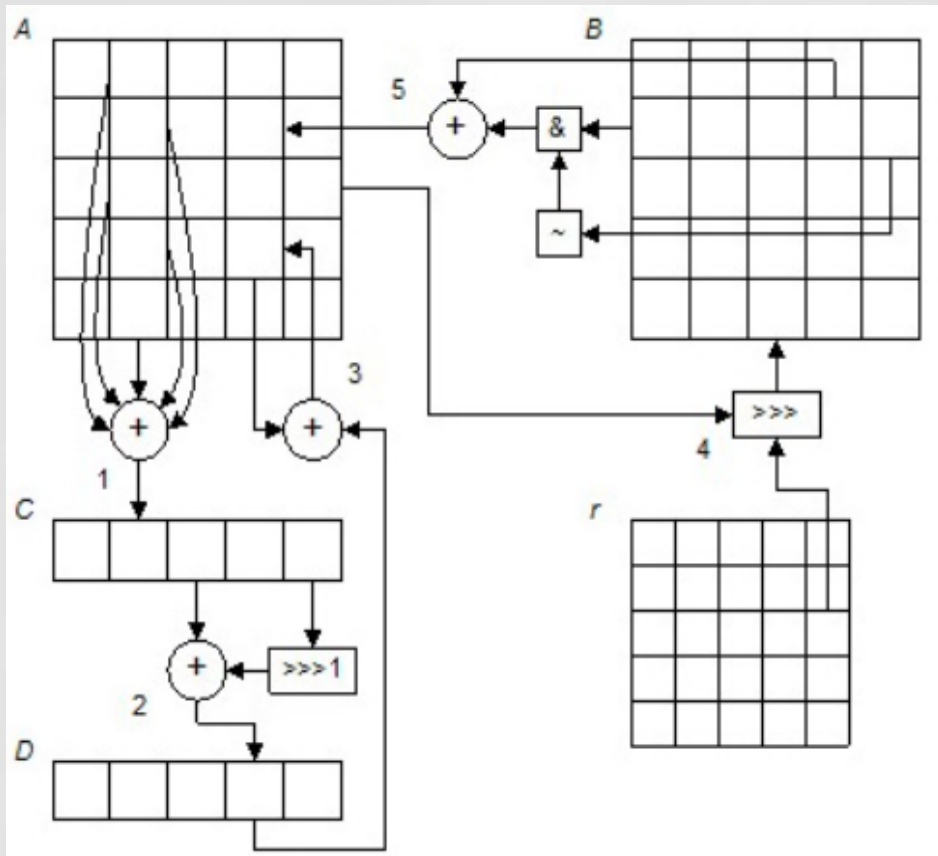
Состояние S – 25×2^l бит ($0 \leq l \leq 6$)
Слои (slices) - (5×5) бит
Проходы (lanes) - 2^l бит
Плоскости, листы, строки,
столбцы и т.п.

- Текущее внутреннее состояние представлено в виде набора битов, сгруппированных в виде виртуального объекта в трёхмерном пространстве (трёхмерного массива)
- Объект можно разбить на плоскости или точнее слои вдоль трёх осей координат, а элементы каждого слоя — на фрагменты в виде столбцов или строк
- Трёхмерное представление текущего внутреннего состояния помогает применить набор простейших логических операций (XOR, AND, NOT) оптимальным образом для реализации псевдослучайных перестановок



Бесключевая псевдослучайная перестановка f

- Раундовые функции выполняют 5-ти шаговую обработку внутреннего состояния
- Количество раундов $12+2^l$ для состояния размера 25×2^l бит ($0 \leq l \leq 6$)



- (1) $C[x] = A[x, 0] \oplus A[x, 1] \oplus A[x, 2] \oplus A[x, 3] \oplus A[x, 4]$, $x = 0 \dots 4$;
- (2) $D[x] = C[x - 1] \oplus (C[x + 1] \ggg 1)$, $x = 0 \dots 4$;
- (3) $A[x, y] = A[x, y] \oplus D[x]$, $x = 0 \dots 4$, $y = 0 \dots 4$;
- (4) $B[y, 2x + 3y] = A[x, y] \ggg r[x, y]$, $x = 0 \dots 4$, $y = 0 \dots 4$;
- (5) $A[x, y] = B[x, y] \oplus (\sim B[x + 1, y] \& B[x + 2, y])$, $x = 0 \dots 4$, $y = 0 \dots 4$,

Где:

B — временный массив, аналогичный по структуре массиву состояния;

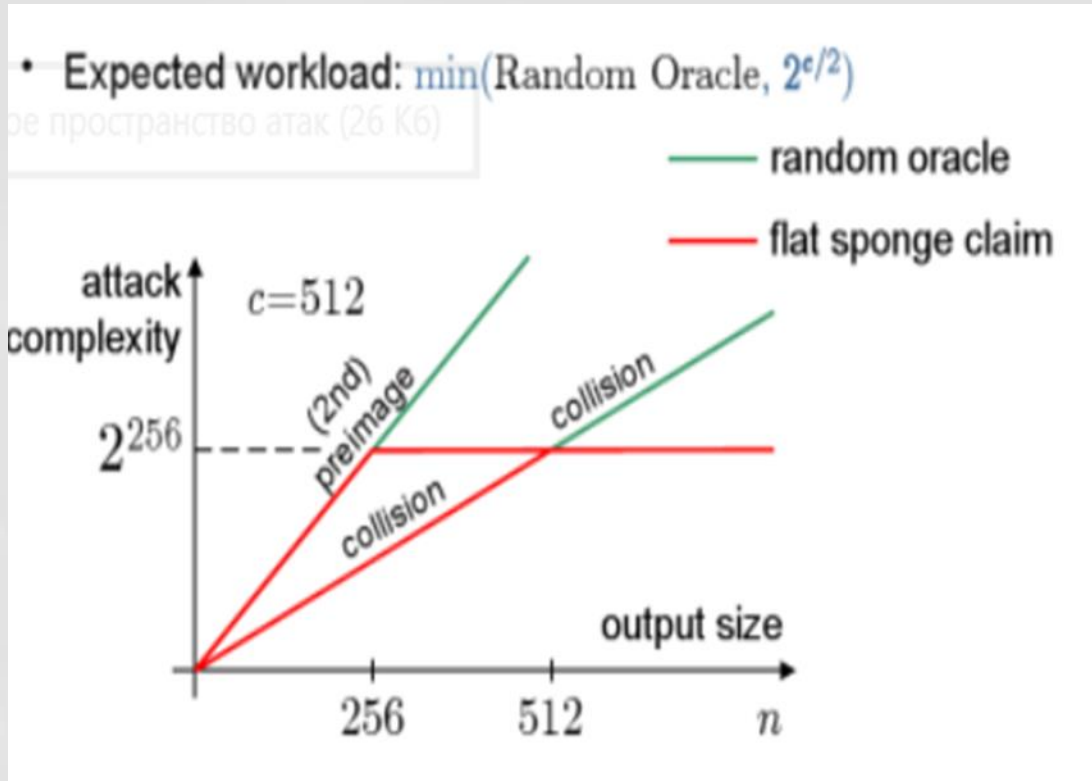
C и D — временные массивы, содержащие по пять 2^l -битных слов;

r — массив, определяющий величину циклического сдвига для каждого слова состояния;

$\sim x$ — **поразрядное дополнение** к x;

операции с индексами массива выполняются по модулю 5.

Оценка криптостойкости



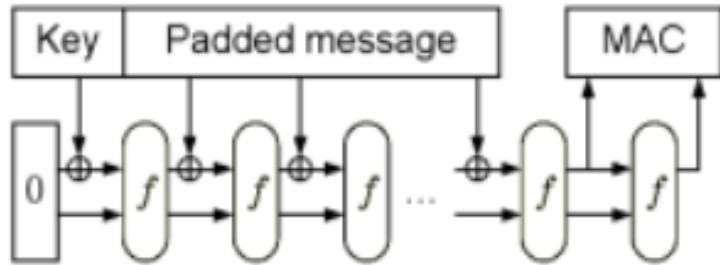
- Авторы Кессак доказали, что стойкость такой конструкции неотличима от стойкости идеальной хэш-функции (*random oracle model*) с размером дайджеста, равным $c/2$ (всё состояние имеет размер $c+r$) при условии, что f — идеальная функция перестановки (без повторений внутреннего состояния)
- Сложность проведения атак гарантируется только до необходимой величины. После увеличения длины выхода хэш-функции за пределы расчетного, стойкость функции перестаёт зависеть от размера выхода

Масштабируемость

- Хэш-функция Кессак реализована таким образом, что функцию перестановки f пользователь может выбирать самостоятельно из набора predetermined функций $\{f-25, f-50, f-100, f-200, f-400, f-800, f-1600\}$
- Каждая функция из набора обрабатывает внутреннее состояние определенного размера $|S| = 25 \times 2^l, 0 \leq l \leq 6$
- Для того чтобы в реализации использовалась функция f обрабатывающая состояния размера $|S|$, необходимо, чтобы $r+c=|S|$
- Количество раундов n применения функции f вычисляется как
$$n = 12 + 2 * l, l = \log_2 |S| / 25$$
- Пример: В качестве стандарта SHA-3 выбрана функция $f-1600$ с размером дайджеста 512 бит. Тогда $c = 2 * 512 = 1024$, размер блока сообщения $r = 1600 - 1024 = 576$ бит, количество раундов $n = 24$ (поскольку $l = 6$)

Кессак , как универсальный криптопримитив

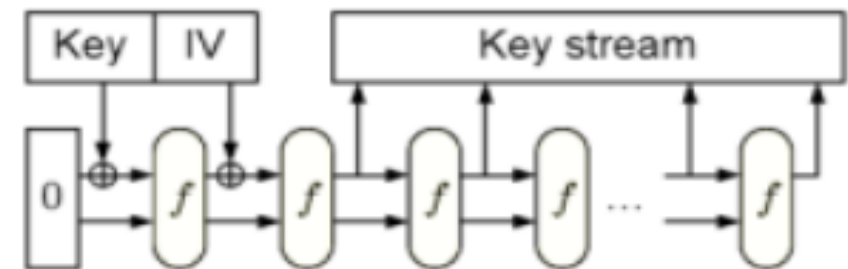
- Message authentication code



- Простое добавление секретного ключа на вход хэш-функции Кессак превращает её в код аутентификации сообщений. Это было невозможно в обычных хэш-функциях SHA-1 или SHA-2 и требовало громоздкой конструкции HMAC

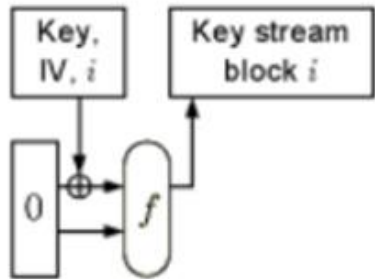
- Добавление секретного ключа с открытым вектором инициализации и вывод гаммы произвольной длины превращает хэш-функции Кессак в потоковый шифр.

- Stream cipher



Кескак, как универсальный криптопримитив

- Random-access stream cipher



- Добавление на вход ключа, вектора инициализации и номера блока позволяет получить потоковый шифр с произвольным доступом — аналог блочного шифра в режиме счётчика (CTR). Это может быть использовано совместно с MAC для шифрования сообщений или пакетов данных

- Использование входа и выхода переменной длины может применяться для генерации симметричных ключей из паролей. Если периодически уничтожать использованные предыдущие состояния, то из такой конструкции легко построить генератор псевдослучайных чисел.

- Mask generating functions, key derivation

