Лекция №4

Коды аутентификации сообщений. Криптографическая хэш-функция

Елена Киршанова

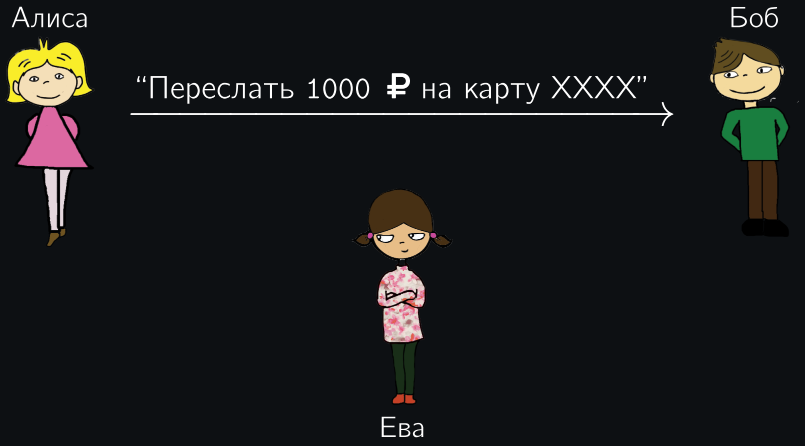
Курс "Основы криптографии"

- В предыдущих лекциях: конфиденциальность сообщений
- В этой лекции: целостность

Криптопритития: Код Аутентификации Сообщения (или Имитовставка)

Message Authentication Code (MAC)

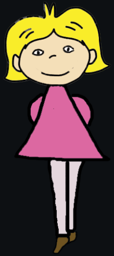
Мотивация



MAC: определение

Цель: отправить сообщение m от Алису к Бобу так, что бы злоумышленник не смог модифицировать m, оставаясь незамеченным

Alice



$k \in \mathcal{K}$

$\mathsf{tag} \leftarrow S(k, m)$

$\xrightarrow{m,\ \mathsf{tag}}$

Bob

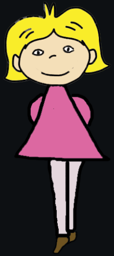
$k \in \mathcal{K}$

$V(k, m, \mathsf{tag}) \in \{0, 1\}$

MAC: определение

Цель: отправить сообщение m от Алису к Бобу так, что бы злоумышленник не смог модифицировать m, оставаясь незамеченным

Alice



$k \in \mathcal{K}$

$\text{tag} \leftarrow S(k, m)$

m, tag

$\longrightarrow$

Bob

$k \in \mathcal{K}$

$V(k, m, \text{tag}) \in \{0, 1\}$

Код Аутентификации Сообщения состоит из 3-х ppt алгоритмов:

- Генерация ключа: $\text{KeyGen}(1^\lambda) : k \leftarrow \mathcal{K}$
- Генерация тага: $S(k, m) : \text{tag} \leftarrow \mathcal{T}$
- Верификация тага: $V(k, m, \text{tag}) : \{0, 1\}$

MAC: корректность и безопасность

Код Аутентификации Сообщения состоит из 3-х ppt алгоритмов:

- Генерация ключа: $\mathsf{KeyGen}(1^\lambda) : k \leftarrow \mathcal{K}$
- Генерация тага: $S(k, m) : \mathsf{tag} \leftarrow \mathcal{T}$
- Верификация тага: $V(k, m, \mathsf{tag}) : \{0, 1\}$

Корректность: $V(k, m, S(k, m)) = 1 \;\forall\; m \in \mathcal{M}, k \in \mathcal{K}$

Безопасность: Для ppt атакующего $\mathcal{A}$, имеющего пары $\{(m_1, t_1), \ldots, (m_N, t_N)\}$ для m_i, выбранных им самим, $\mathcal{A}$ не может сгенерировать новую пару (m, t)

$$(m, t) \notin \{(m_1, t_1), \ldots, (m_N, t_N)\}$$

Поэтому безопасная длина тага: $96, 128, 256$ бит.

MAC: безопасность

$$I = (\text{KeyGen}, S, V) \text{ – MAC}$$

Челленджер $\mathcal{C}$		Атакующий $\mathcal{A}$
$k \leftarrow \text{KeyGen}(1^\lambda)$	$\xrightarrow{\lambda}$	
	$\xleftarrow{m_i}$	$m_i \leftarrow \mathcal{M}$
$t_i \leftarrow S(k, m_i)$	$\xrightarrow{t_i}$	
	$\xleftarrow{(m^\star, t^\star)}$	

$\mathtt{W}_{\mathtt{I},\mathcal{A}}$ – событие $Ver(k, m^\star, t^\star) == 1$.

$\mathtt{MacAdv} = \Pr[\mathtt{W}_{\mathtt{I},\mathcal{A}}]$ – выигрыш $\mathcal{A}$.

Схема MAC I безопасна, если для любого ppt $\mathcal{A}$:

$$\mathtt{MacAdv} = \text{negl}(\lambda).$$

Конструкции MAC

1. Блок-шифр (AES, ГОСТ), псевдослучайная ф-ия f – примеры конструкции MAC для 16-байтных сообщений

$$S(m) := f(k, m) \to t$$
$$V(k, m, t) := f(k, m) == t \; ? \; 1 : 0$$

2. Для более длинных сообщений:

– CBC-MAC (целостность банковских транзакций)
Стандарты: ANSI, FIPS 186-3, ГОСТ

– NMAC (Nested MAC)

– HMAC (SSL, IPSec)

ECBC-MAC

$m = (m_1, m_2, m_3, ...)$, **F**– блок-шифр (AES, ГОСТ), $|m_i|$ – 128 бит

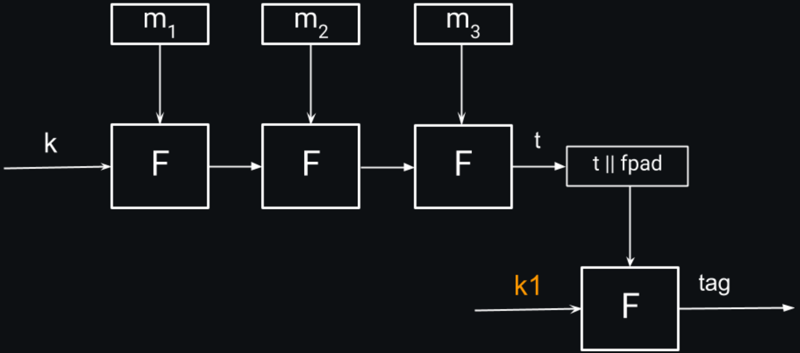
$V((k, k1), m)$ выполняет идентичные шаги.

Отличия от CBC-mode:

- $k, k1$ - два разных ключа для **F**
- Не используется IV
- Выходное значение **tag** может быть урезано (тем самым уменьшая безопасность схемы, стандарты ANSI X9.19, ISO/IEC 9797)

Вложенный (Nested) MAC

$m = (m_1, m_2, m_3, ...)$, **F**– блок-шифр (AES, ГОСТ), $|m_i| - n$ бит, $k \in \{0,1\}^\kappa$.



fpad $\in \{0,1\}^{n-\kappa}$ – любое фиксированное значение

Набивка (padding)

Что если $\mathbf{m} = (m_1, m_2, ...)$ не кратно длине блока?

Пример плохой набивки : добивать $0'$-ми.

$$m = (\star\star\star), \quad |m| < 128\text{бит}$$

Набивка такого m:

$$m_{\text{padded}} = (\star\star\star\, 0 \ldots 0).$$

Тогда для сообщения $m' = (m||0)$:

$$S(k, m_{\text{padded}}) = S(k, m')$$

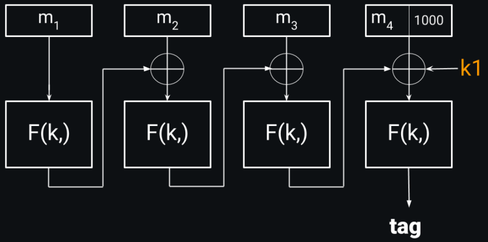
Для $\mathbf{m} \neq \mathbf{m}'$, должно выполнятся $\mathbf{m}||\text{pad} \neq \mathbf{m}'||\text{pad}$.

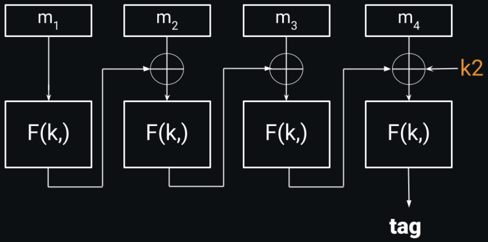
Набивка должна быть функцией $1 \leftrightarrow 1$.

Набивка (padding)

1. ISO. Набивка: $10\dots0$. '1' означает начало набивки.
   Добавляется фиктивный блок, если $|m| <$ длина блока

2. NIST, GOST





$m_4 <$ длина блока

$m_4 =$ длина блока

Преимущества: нет фиктивного блока, нет дополнительного вызова F.

Двухфакторная аутентификация

Двухфакторная аутентификация = пользователь предоставляет то, что он **знает** (пароль) + то, чем он **владеет** (смартфон)

$$I = (\mathsf{KeyGen}, S, V) - \text{ MAC}$$

Пользователь

Сервер

$k \leftarrow \mathsf{KeyGen}()$ $\quad \xrightarrow{k}$

$p = S(k, T)$ $\quad \xrightarrow{p}$

T – текущая дата, время

$V(k, p, T)$

p – временной одноразовый пароль (timed one-time password).

Часть II

Криптографическая хэш-функция

Криптографическая хэш-функция ! = Хэш-функция

Фильтры Блума, контрольные суммы (sumXXX, fletcherXXX), не являются криптографическими хэш-функциями.

Криптографическая хэш-функция: определение

Криптографическая хэш-функция – двойка полиномиальных алгоритмов (Gen, $\mathcal{H}$):

1. Вероятностный Gen : $s \leftarrow \text{Gen}(1^\lambda)$
2. Детерминированный $\mathcal{H}_s : \{0,1\}^\star \rightarrow \{0,1\}^\ell$,

где $\mathcal{H}_s$ является стойкой к коллизиям:

для заданного s, не существует ppt алгоритма, который находит $x, x'(x! = x')$,

$$\mathcal{H}_s(x) = \mathcal{H}(x')$$

Криптографическая хэш-функция обязана быть стойкой к коллизиям.

Существует много коллизий для $\mathcal{H}_s$, но должно быть трудно найти любую коллизию.

Свойство криптографической хэш-функции

$$\mathcal{H}_s : \{0,1\}^\star \to \{0,1\}^\ell$$

I. Стойкость к нахождению прообраза (или односторонность)

Дано: $(s, y \in \{0,1\}^\ell)$
Найти: x, такой что $\mathcal{H}_s(x) = y$

Стойкая к коллизиям хэш-функция является стойкой к нахождению прообраза

II. Стойкость к нахождению 2-го прообраза

Дано: (s, x)
Найти: $x'! = x$, такой что $\mathcal{H}_s(x) = \mathcal{H}_s(x')$

Стойкая к коллизиям хэш-функция является стойкой к нахождению 2-го прообраза

Стойкость к коллизиям $\implies$ II. $\implies$ I.

Экзотические свойства хэш-функций

В мире Bitcoin три свойства хэш-функции могу называться иначе:

нахождение прообраза $\to$ “hiding”

нахождение 2-го прообраза $\to$ “puzzle-friendliness”

стойкость к коллизиям $\to$ “collision resistance”

То есть для реализации Bitcoin подойдет криптографическая хэш-функция.

Атака на любую хэш-функцию: парадокс Дней рождений

Положим $h_1, h_2, \ldots, h_n \in \{0,1\}^\ell$ независимо случайное выбранные строки. Парадокс Дней рождений

$$\text{Для } n = \mathcal{O}\left(\sqrt{|\{0,1\}^\ell|}\right) = \mathcal{O}\left(2^{\ell/2}\right) \quad \Pr[\exists(i! = j) \ : \ h_i = h_j] > 1/2.$$

Алгоритм перебора находит коллизию после $\mathcal{O}(2^{\ell/2})$ вычисленных хэшей:

1. Выбрать $2^{\ell/2}$ случайных строк $m_1, \ldots, m_{2^{\ell/2}}$
2. Для каждой m_i вычислить $h_i = \mathcal{H}_s(m_i)$, отсортировать пары (h_i, m_i) по значению h_i
3. Найти в упорядоченном списке $h_i = h_j$. Коллизия: (m_i, m_j).

Алгоритм успешен с константной вероятностью по парадоксу ДР.

Вывод: Требуем $\ell \geq 160$.

Хэш-функции: исторический дайджест

1. 1980е: MD4 (Message Digest) предложено R. Rivest. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
2. 1990: MD5. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд

Хэш-функции: исторический дайджест

1. 1980е: MD4 (Message Digest) предложено R. Rivest. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
2. 1990: MD5. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Статус: Взломана. См. `https://shattered.io/` два PDF файла с одинаковым значением SHA-1.
   !: всё еще используется некоторыми системами (GIT).

Хэш-функции: исторический дайджест

1. 1980е: MD4 (Message Digest) предложено R. Rivest. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
2. 1990: MD5. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Статус: Взломана. См. `https://shattered.io/` два PDF файла с одинаковым значением SHA-1.
   !: всё еще используется некоторыми системами (GIT).
4. 2001: SHA-2 (SHA-256, SHA-384, SHA-512). $\ell = 256, 348, 512$
   Статус: Считается безопасной.

Хэш-функции: исторический дайджест

1. 1980е: MD4 (Message Digest) предложено R. Rivest. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
2. 1990: MD5. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Статус: Взломана. См. `https://shattered.io/` два PDF файла с одинаковым значением SHA-1.
   !: всё еще используется некоторыми системами (GIT).
4. 2001: SHA-2 (SHA-256, SHA-384, SHA-512). $\ell = 256, 348, 512$
   Статус: Считается безопасной.
5. 2012: SHA-3 (Keccak). SHA-3 $\ell = 224/256/348/512.$
   Статус: Считается безопасной.

Хэш-функции: исторический дайджест

1. 1980е: MD4 (Message Digest) предложено R. Rivest. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
2. 1990: MD5. $\ell = 128$
   Статус: Взломана. Коллизию можно найти в течение секунд
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Статус: Взломана. См. `https://shattered.io/` два PDF файла с одинаковым значением SHA-1.
   !: всё еще используется некоторыми системами (GIT).
4. 2001: SHA-2 (SHA-256, SHA-384, SHA-512). $\ell = 256, 348, 512$
   Статус: Считается безопасной.
5. 2012: SHA-3 (Keccak). SHA-3 $\ell = 224/256/348/512.$
   Статус: Считается безопасной.

В России:

1. GOST R 34.11-94 and GOST 34.311-95. $\ell = 256$
   Статус: Считается устаревшей. Коллизия за 2^{105} операций,
2. GOST R 34.11-2012. Стрибог $\ell = 256, 512$
   Статус: Считается безопасной.

Часть III

Конструкция Меркла-Дамгора

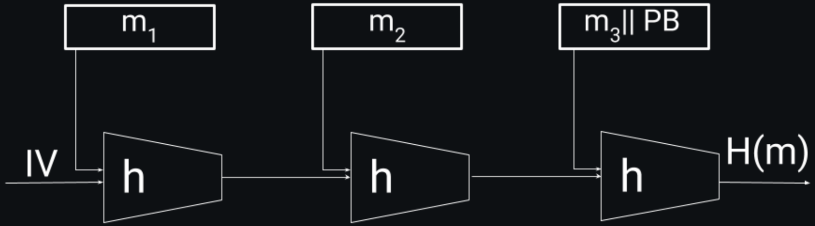
Конструкция хэш-функций: парадигма Merkle-Damgård

Из функции компрессии (определим позже)

$$h : \mathcal{K} \times \mathcal{M} \to \mathcal{K}$$

построим $\mathcal{H} : \mathcal{M}^\star \to \mathcal{K}$.

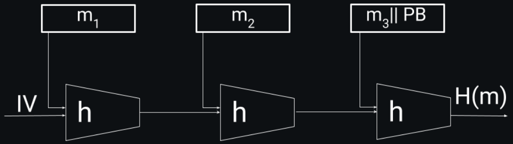
Пусть $m = (m_1, m_2, m_3)$ произвольной длины



IV – Начальное значение, фиксировано для конкретной хэш-функции
PB – Блок добивки $[100\ldots0||\,|m|\,]$.
Если **PB** не влезает, добавляем новый блок.

Безопасность конструкции Merkle-Damgård

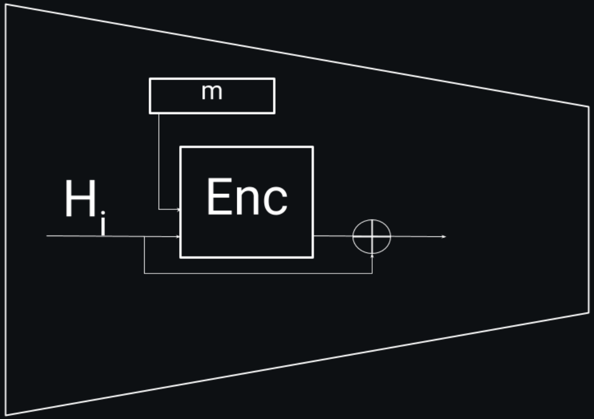
Теорема: Если h стойкая к коллизиям, то и H стойкая к коллизиям.



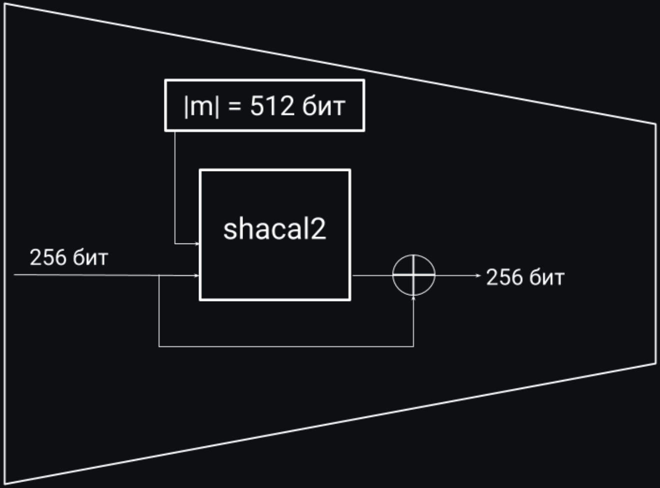
Конструкции функции компрессии h

$$\mathsf{Enc} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n \text{ – блок-шифр.}$$

Конструкция Davies-Meyer: $h(H_i, m) = \mathsf{Enc}(H_i, m) \oplus H_i$.

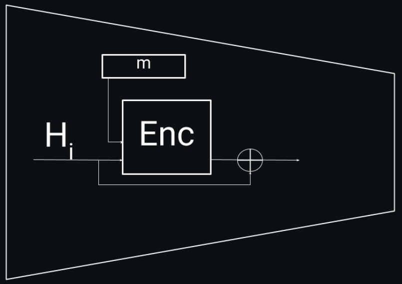


Пример: Функция компрессии в SHA-256



Безопасность Davies-Meyer

Теорема (неформально):
$\mathsf{Enc}(k, \cdot) : \{0,1\}^n \to \{0,1\}^n$ неотличима от случайной перестановки, то нахождение коллизии $h(H, m) = h(H', m')$ требует $\mathcal{O}(2^{n/2})$ вычислений $(\mathsf{Enc}, \mathsf{Enc}^{-1})$.



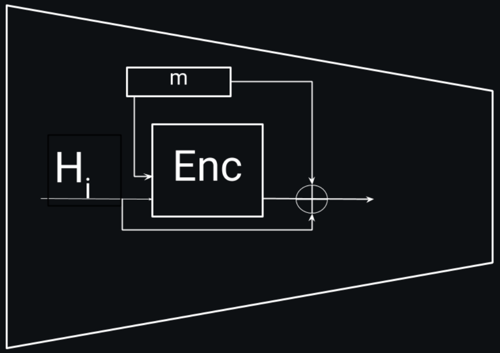
Альтернативные конструкции h

Конструкция Davies-Meyer:

$$h(H, m) = \mathsf{Enc}(H, m) \oplus H$$

Конструкция Miyaguchi–Preneel:

$$h(H, m) = \mathsf{Enc}(H, m) \oplus H \oplus m$$



ГОСТ Р 34.11-2012 (Стрибог) использует Miyaguchi–Preneel.

Другие комбинации Enc, H, m возможны, см. B. Preneel, R. Govaerts, J. Vandewalle. “Hash functions based on block ciphers: a synthetic approach.”

Не все комбинации безопасны!

Часть IV

Где используются хэш-функции

- Построение MACa
- Протокол идентификации
- Доказательство работы (proof of work)

1. Построение МАСа

Построение МАСа

Задача: из хэш-функции

$$H : \{0,1\}^\star \to \{0,1\}^\ell$$

построить функцию генерации МАСа

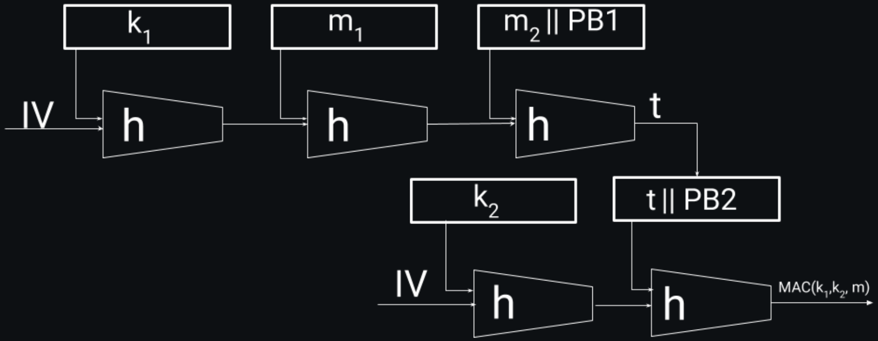
$$S : \mathcal{K} \times \{0,1\}^\star \to \{0,1\}^\ell.$$

Основная сложность: хэш-функция – бесключевой примитив.

Конструкция “Two-key Nest”

Для сообщения $M = (m_1, m_2)$ и $h : \{0,1\}^\ell \times \mathcal{M} \to \{0,1\}^\ell$ – функции компрессии

$$S((k_1, k_2), M) = \mathcal{H}_{\mathsf{NMAC}}(k_2 \mid\mid \mathcal{H}_{\mathsf{NMAC}}(k_1 \mid\mid M)$$



Теорема: Если $h(\cdot,)$ и $h(, \cdot)$ – псевдослучайная функция, то Two-key Nest – безопасный MAC.

2. Протокол идентификации

Протокол идентификации: определение

$\mathsf{Id} = (\mathsf{KeyGen}, \mathsf{P}, \mathsf{V})$ – интерактивный протокол, состоящий из

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$
- $\mathsf{P}(\mathsf{sk})$ – Доказывающий (Prover)
- $\mathsf{V}(\mathsf{vk}) \to \{0, 1\}$ – Проверяющий (Verifier)

Корректность: $\mathsf{V}(\mathsf{vk}) \to 1$ при общении с sk, где $\mathsf{KeyGen}(1^\lambda) = (\mathsf{vk}, \mathsf{sk})$.

Безопасность:

Челленджер $\mathcal{C}$		$\mathcal{A}$
$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$	$\xrightarrow{\mathsf{vk}}$	
	$\leftarrow$	
	$\rightarrow$	

Выигрыш $\mathcal{A}$ – $\mathcal{C}$ возвращает "1" (Accept).

Id – безопасный Id-протокол относительно прямых атак,
если $\forall$ ppt $\mathcal{A}$ вероятность выигрыша $\mathrm{negl}(\lambda)$.

Протокол парольной идентификации

$$\mathcal{H} : \{0,1\}^\star \to \{0,1\}^\ell - \text{ хэш-функция}$$

$$\mathsf{KeyGen} \to (\mathsf{sk} = pwd, \mathsf{vk} = \mathcal{H}(pwd))$$

$\mathsf{P}(pwd)$ $\qquad\qquad\qquad$ $\mathsf{V}(\mathsf{vk})$

$$\xrightarrow{\quad pwd \quad}$$

$$\mathcal{H}(pwd) == \mathsf{vk} \ ?\ 1 \ :\ 0$$

$\mathcal{H}$ – криптографическая хэш-функция $\implies$ протокол $\mathsf{Id} = (\mathsf{KeyGen}, \mathsf{P}, \mathsf{V})$ безопасен относительно прямых атак.

Протокол парольной идентификации

$$\mathcal{H} : \{0,1\}^\star \to \{0,1\}^\ell - \text{ хэш-функция}$$

$$\mathsf{KeyGen} \to (\mathsf{sk} = pwd, \mathsf{vk} = [\mathcal{H}(pwd, salt), salt \xleftarrow{\$} S])$$

$\mathsf{P}(pwd)$ $\qquad\qquad$ $\mathsf{V}(\mathsf{vk} = (h, salt))$

$$\xrightarrow{\quad pwd \quad}$$

$$\mathcal{H}(pwd, salt) == h \ ?\ 1 \ :\ 0$$

$\mathcal{H}$ – криптографическая хэш-функция $\implies$ протокол $\mathsf{Id} = (\mathsf{KeyGen}, \mathsf{P}, \mathsf{V})$ безопасен относительно прямых атак.

3. Доказательство работы (proof of work)

Хэш-функция в BitCoin

Важный примитив в BitCoin: Proof of Work (PoW) / Доказательство работы

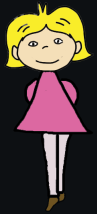
Интуиция: вычислительная мощность пользователя $\implies$ пользователь должен доказать это результатом вычислений

- PoW предложен Dwork & Naor (1992) как противодействие спаму
- Идея: заставить пользователя решить пазл "средней сложности" (решение должно быть легко верифицировать)

Хэш-функция в BitCoin: конструкция PoW

Основной примитив: криптографическая хэш-функция $\mathcal{H} : \{0,1\}^\star \to \{0,1\}^\ell$, сложность вычисления которой $T(\mathcal{H})$.

Алиса
Доказывающий



Боб
Проверяющий

1. $x \in \{0,1\}^\star$

$\xleftarrow{\;x\;}$

2. Выбрать $s \in \{0,1\}^\star$
т.ч. $\mathcal{H}(s||x)$
начинается с n 0'ей

Время: $2^n T(\mathcal{H})$

$\xrightarrow{\;s\;}$

3. Проверить:
$\mathcal{H}(s||x)$ имеет n 0ей?

Время: $T(\mathcal{H})$

Для криптографической хэш-функции $\mathcal{H}$ Алиса не может найти s быстрее, чем перебором. Это атака на прообраз.