Специальные разделы криптологии

Савчук М.Н.

22 апреля 2015 г.

Глава 1

Алгоритм Шора для факторизации

1.1 Введение

Есть число N

$$N = a \cdot b, \qquad a \neq b$$

Задача факторизации: для числа $N = p_1^{\alpha_1} \cdots p_t^{\alpha_t}$

- либо получить каноническую форму
- либо получить нетривиальный делитель

Число $a \in \mathbb{Z}_N$ **принадлежит показателю** δ, если δ — минимальное число, для которого $a^\delta \equiv 1 \mod n$. Обозначаем $\delta_N(a)$.

Показатель существует только если a и N взаимно просты ($a \in \mathbb{Z}_N$) и по сути является порядком a в группе $\mathbb{Z}_N : \delta_n(a) = ord_N(a)$. Если $gcd(a, N) \neq 1$, то мы нашли нетривиальный делитель.

Допустим, что для заданного $a \in \mathbb{Z}_N$ у нас есть оракул O_f, который возвращает показатель числа. Если этот оракул полиномиален, то можно факторизовать число (вероятностно)

Пусть r — чётное и $gcd\left(a^{r/2} + 1, N\right) = 1$, где $O_f(a) = r$

$$a^r \geq 1 \mod N$$

$$\left(a^{r/2} - 1\right) \cdot \left(a^{r/2} + 1\right) \equiv 0 \mod N$$

Но $a^{r/2} \neq 1 \mod N$, потому что иначе r — не минимальное число (нарушение определения показателя). Значит, $\left(a^{r/2} - 1\right)$ и N имеет нетривиальный общий делитель.

Важно: условие $gcd\left(a^{r/2} + 1, N\right) = 1$ можно заменить на условие $a^{r/2} + 1 \nmid N$.

Утверждение 1.1.1. *Пусть* $N = p_1^{\alpha_1} \cdots p_t^{\alpha_t}$ *и*

$$S = \left\{a \in \mathbb{Z}_N : ord(a)\, mod 2 = 0 \vee a^{ord(a)/2} + 1 \nmid N\right\}$$

Тогда

$$|S| \leq \frac{\varphi(N)}{2^k}$$

То есть, подходящих нам a очень мало.

1.2 Алгоритм факторизации с оракулом

1. Генерируем такое a, чтобы при $r = O_f(a)$ выполняясь

$$r = 2 \cdot r_1, a^{r_\alpha} + 1 \nmid N$$

2. $gcd\left(a^{r_1} - 1, N\right)$ — нетривиальный делитель.

Утверждение 1.2.1. *В классической модели найти оракул O_f не получилось. Шору удалось построить его в квантовой модели.*

Рассмотрим функцию $f : \mathbb{Z}_N \to \mathbb{Z}_N$

$$f(x) = a^x \mod N$$

Это почти непериодическая функция — её период не кратен N.

1.3 Квантовая система

У нас есть $|0\rangle$, $|1\rangle$; кубит находится в суперпозиции состояний $\alpha_1 \cdot |0\rangle + \alpha_2 |1\rangle$, над которыми можно выполнять унитарные операции в пространстве Гильберта $C \cdot \mathbb{Z}_2$.

Коэффициенты α_1, α_2:

- $\alpha_1, \alpha_2 \in \mathbb{C}$,
- $|\alpha_1|^2 + |\alpha_2|^2 = 1$,
- $|\alpha_1|^2$ — вероятность попасть в $|0\rangle$,
- $|\alpha_2|^2$ — вероятность попасть в $|1\rangle$.

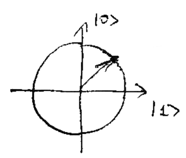


Рис. 1.1: Состояние кубита

К кубитам можно применять преобразование Уолша-Адамара

$$W(|0\rangle) = \frac{1}{\sqrt{2}} \cdot (|0\rangle + |1\rangle)$$
$$W(|1\rangle) = \frac{1}{\sqrt{2}} \cdot (|0\rangle - |1\rangle)$$

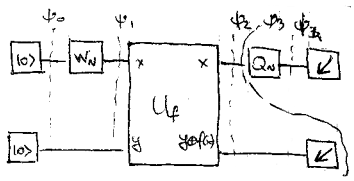


Рис. 1.2: Алгоритм Шора

Набор кубитов — одна из возможных интерпретаций квантово-механической системы. Другой вариант — N-уровневая система, в которое есть N состояний $|0\rangle$, $|1\rangle$, $\dots$, $|N-1\rangle$, и эти состояния ортонормированы (то есть, при измерении выпадают только эти состояния и ничего среднего между ними).

Обозначения на рис. 1.2:

- ψ_i — состояние системы,
- W_N — преобразование Уолша (или преобразование Фурье),
- U_f — стандартный оракул (базисный в квантовой модели),
- Q_N — преобразование Фурье,
- стрелочка — измерение.

Что происходит?

1. $$|\psi_0\rangle = |0\rangle\,|0\rangle$$

2. $$|\psi_1\rangle = W_N\left(|0\rangle\right) \otimes |0\rangle$$

   Свойство: $W_N\left(|0\rangle\right)$ даёт равномерную суперпозицию (что Уолш, что Фурье), то есть

   $$W_N\left(|0\rangle\right) = \frac{1}{\sqrt{N}} \cdot \left(|0\rangle + |1\rangle + \dots + |N-1\rangle\right)$$

3. Суперпозиция всех значений функции $f(x) = a^x \mod N$

   $$|\psi_2\rangle = \frac{1}{\sqrt{N}} \cdot \left(|0\rangle \cdot \left|a^0 \mod N\right\rangle + |1\rangle \cdot \left|a^1 \mod N\right\rangle + \dots + |N-1\rangle \cdot \left|a^{N-1}\right\rangle \mod N\right)$$

4. Мы измерили второй регистр – там второй кубит принял некоторое значение $|y_0\rangle$

   $$|\psi_0\rangle = \frac{1}{\sqrt{k}} \cdot \left(|x_0\rangle + |x_0 + r\rangle + \dots + |x_0 + (k-1)\cdot r\rangle\right) \cdot |y_0\rangle\,,$$

   где $|x_0\rangle$ — все состояния $|x\rangle$, для которых $f(x) = y_0$, $k = \lfloor \frac{N}{r} \rfloor$.

5. Преобразование Фурье

$$Q_N\left(|k\rangle\right) = \frac{1}{\sqrt{N}} \cdot \sum_t \exp\frac{2\cdot\pi\cdot i}{N} \cdot k \cdot t \cdot |t\rangle$$

Применяя его к первому регистру, получаем

$$|\psi_4\rangle = \sum_S c\left(S\right) \cdot |S\rangle \cdot |y_0\rangle\,,$$

где

$$c\left(S\right) = \begin{cases} \dfrac{\sqrt{N}}{r} & , S \equiv 0 \mod \dfrac{N}{r}, \\ 0 & , S \not\equiv 0 \mod \dfrac{N}{r}. \end{cases}$$

Таким образом это будет или 0, или равномерная суперпозиция.

6. Измеряем первый регистр — можем получить тоьлко те S, у которых $c\left(S\right) \neq 0$, т.е. состояния вида $\left|\beta \cdot \dfrac{N}{r}\right\rangle$, где β — неизвестная константа.

Таким образом мы знаем какое-то число S такое, что

$$S = \beta \cdot \frac{N}{r}.$$

Если $gcd\left(S, N\right) = 1$ и $N \mid r$, то всё, но у нас $N \nmid r$! Поэтому на самом деле

$$\frac{S}{N} = \frac{\beta}{r},$$

и мы строим рациональные приближения при помощи цепных дробей.

Итоговая сложность алгоритма Шора — $O\left(\log^3 N\right)$.

Глава 2

Задача о скрытой подгруппе

Глава 3

Современные хэш-функции

3.1 Подходы к построению сжимающей функции

Сжимающие функции современных хэш-функций строятся:

- при помощи блочных шифров (см. ниже)
- при помощи поточных шифров — в первую очередь, из-за скорости работы; обычно состояние хэша отождествляется с состоянием поточного шифра
- при помощи алгебраических операций (modular hashing) — вызвано массовым распостранением аппаратуры, заточенной под RSA, и, соответственно, доступностью умножения и возведения в степень по модулю
- на NP-задачах (задачи на решётках, задачи укладки рюкзака ...)
- при помощи всякой экзотики (динамических хаос, клеточные автоматы, ...)

3.1.1 Схемы на основе блочных шифров

Рассмотрим блочный шифр

$$E : \{0,1\}^m \times \mathcal{K} \to \{0,1\}^m$$

$$Y = E_k(X)$$

Функция Матяша-Мейера-Озиса (Matyas-Meyer-Oseas)

$$H_i = E_{k(H_{i-1})}(X_i) \oplus X_i$$

тут $k(H_i)$ — преобразование блока H_i в ключ шифрования

Считается, что эта функция необратима, но это не доказано. Более того, непонятно даже, какие требования нужно выдвигать к шифру E, чтобы достигалась стойкость.

Однако Пренель доказал, что можно построить 12 сжимающих функций, эквивалентных по стойкости функции MMO. Две главные из них:

Функция Дэвиса-Мейера (Davis-Mayer)
$H_i = E_{k(X_i)}(H_{i-1}) \oplus H_{i-1}$

Функция Миягути-Пренеля (Miyaguchi-Preneel)
$H_i = E_{k(H_{i-1})}(X_i) \oplus X_i \oplus H_{i-1}$

Все указанные функции неявно предполагают, что n = m = ω, т.е., что размеры блока, состояния и хэша совпадают. Однако размеры блоков современных шифров (64 и 128 бит) недостаточны для обеспечения стойкости хэш-функции к атакам общего вида.

Чтобы повысить стойкость и увеличить длину хэша по отношению к размеру блока шифра, используют более навороченные схемы.

Функция MDC-2

— разработана Брахтом(Bracht), носит также название функции Мейера-Шиллинга (Meyer-Shilling)
— включена в ISO/IEC 10118-2 как рекомендованная схема

Функция сжатия в MDC-2 имеет вид:

$$(H_i, \widetilde{H}_i) = f(H_{i-1}, \widetilde{H}_{i-1}, X_i)$$

и использует 2 стартовых вектора $H_0 = IV, \widetilde{H}_0 = \widetilde{IV}$

Хэш-функция MDC-2 = схема MD + описанная функция сжатия.

Самая лучшая из известных атак на MDC-2 требует $O(2^{3m})$ операция для поиска прообраза и $O(2^m)$ для поиска коллизии.*

Однако собственно сжимающая функция в этом плане слабая: требуется $O(2^m)$ и $O(2^{m/2})$ операция соответственно.

Этой слабости лишена функция MDC-4, но она ещё более навороченная, использует 4 шифра за раунд и потому ещё медленнее.

3.1.2 Обобщения схемы Меркле-Дамгора

Придумано множество вариантов схемы MD, которые убирают те или иные конструктивные недостатки или защищают от некоторых атак. Рассмотрим основные из них.

*m — размер блока шифра, а не хэша

Концепция wide-pipe ("широкого канала")

Предложена Люксом (Lucks).

Идея проста: раз уж стойкость хэш-функции сводится к стойкости сжимающей функции, необходимо сделать часть состояния хэша ненаблюдаемой: $\omega \geq 2n$. Тогда поиск коллизии для сжимающей функции требует $O(2^{\omega/2}) = O(2^n)$ операций — столько же, сколько и поиск прообраза. Канал широкий, в конце резко сужается.

Функции с $\omega = n$ стали называть narrow-pipe ("узкий канал")

Рандомизированное хэширование

Превращает хэш-функцию в универсальную хэш-функцию. Используется дополнительный параметр — "соль"(salt), который должен быть случайным.

$$H_0 = IV; r = random$$

$$H_i = f(H_{i-1}, X_i \oplus r), i = \overline{1, t}$$

$$h(X) = (g(H_t), r)$$

Рандомизация помогает защититься от атак на основе предвычислений (атаки компромисса), поскольку значение функции от одинаковых входов будет всякий раз разное. Стойкость сильно зависит от стойкости используемого ГСЧ.

Схема HAIFA (HAsh Iterated FrAmework)

На каждом шаге используется 2 дополнительных параметра:

s — salt, обеспечивает параметризацию (или рандомизацию); если параметризация не нужна, то $s = 0$;

l_i — длина той части входного сообщения, которая обработана на данный момент, включая блок X_i (но без учёта паддинга)

Соответственно,

$$H_i = f(H_{i-1}, X_i, s, l_i)$$

Такое простое изменение резко усложнит поиск коллизий за счёт уникальности обработки каждого блока данных X_i в зависимости от позиции i. † Недостаток — потеря в вычислительной эффективности (f большого размера и, соответственно, её вычисление требует больших ресурсов).

На схеме HAIFA построены хэш-функции Blake (финалист SHA-3) и "Стрибог"(стандарт РФ)

Хэш-функция Skein

Один из финалистов конкурса SHA-3, отличается дерзким дизайном :)

- используется специально разработанный шифр TreeFish
- ARX-дизайн: в вычислениях используются только сложение, xor и циклические сдвиги, что приводит к бешенной скорости работы
- заточка под 64-битную архитектуру

† Аналогия с полиалфавитными шифрами

- narrow-pipe (в отличие от всех прочих финалистов SHA-3)
- специальный режим работы блочного шифра для хэширования

Режим UBI (Unique Block Iteration) Главная цель как в и HAIFA — сделать сжатие каждого блока как можно более уникальным

$$H_i = E_{k(H_{i-1})}(format(x_i, cfg))$$

тут format — функция форматирования входных данных в битовый блок по размеру входа шифра
cfg — служебные данные, которые включают в себя:

- длину обработанного сообщения в байтах(на данный момент)
- флаг "это первый блок"(1 бит)
- флаг "это последний блок"(1 бит)
- флаг "bitpad"(1 бит) — использовалось ли выравнивание в битах
- идентификатор типа хэша (примерно равно salt)
- и всё, что пожелает разработчик

Схема Grøstl

Хэш-функция Grøstl — тоже финалист SHA-3

- радикальный wide-pipe
- отказ от использования шифров как более уязвимых к атакам на основе неподвижных точек и rebound-атак

Внутреннее состояние $\omega \geq 2n$; pad(X) включает длину сообщения. Длина блока сообщения $m = \omega$; используются две перестановки P и Q над $\{0,1\}^\omega$, которые должны быть "существенно различны"

$H_0 = IV(n)$ (стартовый вектор содержит длину хэша)
$H_i = f(H_{i-1}, X_i), i = \overline{1,t}$
$f(h,m) = P(h \oplus m) \oplus Q(m) \oplus h$
$h(x) = g(H_t)$
$g(h) = trunc_n(P(h) \oplus h)$

Из всех финалистов SHA-3 Grøstl — самая консервативная (и надёжная в этом плане) хэш-функция. Недостаток — и самая медленная, поскольку требуеются две огромные перестановки (в самом Grøstl они реализованы на AES-подобных преобразованиях).

Новый украинский стандарт хэш-функции ДСТУ 7564:2014 ("Купина") построен на схеме Grøstl

Схема Sponge ("губка")

На схеме Sponge основан алгоритм Кессак — победитель SHA-3

- полный отказ от всех существовавших архитектурных решений
- ориентация на битовые архитектуры
- wide-pipe
- использование только простых преобразований (без S-блоков)
- переменная длина генерируемого хэша (потенциально до бесконечности)
- не требует включения длины сообщения во входной поток (pad)

Состояние H разбивается на две части: R (rate — "такса") — m бит, и C (capacity — "ёмкость") — c бит; $\omega = m + c$

$$H_0 = R_0||C_0 = 0^\omega$$
$$H_i = R_i||C_i = f(R_{i-1} \oplus X_i||C_i), i = \overline{1,t}$$
$$Z_{i-t} = R_i - 1$$
$$R_i||C_i = f(R_{i-1}||C_{i-1})$$
$$h(X) = trunc_n(Z_1||Z_2||\ldots)$$

Свойства:

- Для Sponge используется фиксированная перестановка f над $\{0,1\}^\omega$. В силу сложности построения такой перестановки ω обычно тоже фиксируют
- Чем больше m, тем больше хэш; чем больше C, теб он надёжнее. Разработчики рекомендуют брать $c = 2n$ и $m = \omega - c$. ‡
- Единственное требование к паддингу — входной поток не должен заканчиваться на серию нулей (иначе, как видно из схемы, возможны атаки на основе удлинения сообщений). Поэтому разработчики предложили sponge-padding вида pad$(X) = 10\ldots01$

Поскольку Sponge генерирует хэш переменной длины, то очевидно, что стандартные методы оценивания стойкости к атакам общего вида (поиск прообраза, дни рождения и т.д.) не подходят Оказывается, для Sponge параметром стойкости является не длина хэша, а ёмкость.

Теорема. Для Sponge атаки общего вида требуют:
— $O(2^c)$ операций для поиска (второго) прообраза
— $O(2^{c/2})$ операций для поиска коллизии

Показано, что стойкость Sponge сводится к сложности решения задачи CICO (constrained input, constrained output):
пусть $f : \{0,1\}^N \to \{0,1\}^N$ — биекция, и $X, Y \subseteq \{0,1\}^N$;
требуется определить, существует ли $x \in X$ такой, что $f(x) \in Y$.

В общем случае эта задача считается сложной ($\in$ NP). Перестановка f должна выбираться с учётом требования стойкости к этой задаче.

‡в Keccak $\omega = 1600, c = 2n$, где $n = 128\ldots512$

Глава 4

Коды аутентичности сообщений

Код аутентичности сообщения или имитовставка (message authentication code, MAC) — криптопримитив, направленный на защиту целостности конфиденциальных данных, точнее, выявления их изменений. MAC-коды аналогичны электронным цифровым подписям, но относятся к классу симметричных алгоритмов (т.е., для вычисления и проверки имитовставки требуется один ключ), а потому не обеспечивают аутентификациюю отправителя — только аутентификацию данных.

С математической точки зрения MAC-код — это отображение вида

$$mac : \{0,1\}^{l(n)} \times K \to \{0,1\}^n ,$$

где $k \in K$ — секретный параметр (ключ), которое удовлетворяет условиям стойкости к атакам.

4.1 Атаки на MAC-коды

- Восстановление ключа (key recovery) — по известному множеству

  $$(x_i, mac_k(x_i)), i = \overline{1, N}$$

  восстановить значение ключа k или его части. При восстановлении ключа аналитик может в дальнейшем подделывать любую подпись

- Подделка (forgery) — по известному множеству $(x_i, mac_k(x_i)), i = \overline{1, N}$ вычислить правильное значение $y = mac_k(x')$ для некоторого $x' \notin \{x_i\}$

Формальные определения стойкости к этим атакам довольно сложны и оперируют тремя параметрами: время работы противника T, вероятность успеха ε и размер обучающей выборки N. Мы их не приводим.

4.2 Подходы к построению MAC-кодов

- На основе готовых хэш-функций

- На основе специальных режимов работы блочных шифров
- На основе итеративных конструкций типа схемы Меркле-Дамгора — в первую очередь схема Шупа (Shoup)
- На основе комбинаторных универсальных хэш-функций. Подход уникальный тем, что определение стойкости не будет зависеть от модели вычислений; в прочих аспектах несколько противоречивый.

4.3 Атака расширения длины (length extension attack)

Применима к классическим схемам Меркле-Дамгора (с $n = \omega$ и $g(x) = x$). К этому типу хэшей относятся $MD5$, $SHA-1$, $SHA-2$, $RIPEMD$ — то есть все популярные хэш-функции конца XX века.

Пусть x — сообщение и $y = mac_k(x) = h(k||x)$. Атакующий формирует сообщение $x' = x||pad(x)||z, \forall z$ и вычисляет $y' = mac_k(x')$ очень простым образом: $y' = h(z)$, если вместо стартового вектора IV взять значение y.

4.4 Схемы на основе хэш-функций

Мы используем хэш-функцию $h(x)$.

Рассмотрим сначала несколько неочевидно плохих схем:

1. $mac_k(x) = h(k||x)$

   Эта схема подвержена атакам на структуру итеративной цепи у хэш-функции, которые отталкиваются от того, что ключ фактически только задаёт стартовое состояние для вычисления хэша от x.

2. $mac_k(x) = h(x||k)$

   Эта схема оказалась плоха тем, что ключ влияет только на последние стадии вычисления. Значит, если находится коллизия на X, она тут же превращается в forgery для MAC.

3. $mac_k(x) = h(k_1||x||k_2), k = (k_1, k_2)$

   Эта схема после тщательного анализа тоже оказалась с уязвимостями что при $k_1 = k_2$, что при $k_1 \neq k_2$.

4.5 Функция HMAC

Определена стандартом RFS 2104 “Keyed-Hashing for Message Authentication” и включена в ISO.

Как и раньше, n — длина хэша, m — размер блока, который обрабатывается хэш-функцией $h(x)$.

Ключ k может быть любой длины, но рекомендуется не меньше n бит. Используется две константы:

- $ipad = 0x363636\ldots36$ — байты $0x36$ повторенные столько раз, чтобы $|ipad| = m$

- $opad = 0x5C5C5C\ldots 5C$ — байты $0x5C$ повторенные столько раз, чтобы $|opad| = m$

Вообще говоря, константы могут быть любыми; тут они выбраны так, чтобы покрыть все биты и по возможности не повторяться.

Вычисление $HMAC$ происходит в два этапа:

1. Выравнивание ключа:

   - если $|k| < m$, ключ дополняется нулевыми байтами до длины m
   - если $|k| > m$, то $k = h(k)$, после чего дополняется нулями до длины m

   получаем ключ k, $|k| = m$.

2. Вычисление

$$HMAL_k(x) = h(k \oplus ||h(k \oplus ipad||x))$$

За счёт того, что используется двойное хэширование и результат первого недоступен противнику, атаки типа расширения длины перестают работать.

В целом HMAC состоит из одних дотоинств и ровно одного недостатка: скорость работы (хэши медленные, а тут ещё и два раза). Блочные шифры в плане скорости выглядят перспективнее.

Глава 5

Математическая теория кодов аутентификации

Сюда мы включаем и цифровые подписи и коды–имитовставки. Стойкость кодов аутентификации базируется на:

- вычислительной стойкости (это очень очень сложно)
- доказуемая стойкость (сведение к признанно сложной задаче)
- безусловная стойкость (недостаточно информации для взлома)

Участники процесса аутентификации:

- отправитель A (Алиса)
- получатель B (Боб) (A и B могут быть одним лицом)
- криптоаналитик Kr (Крамер)
- арбитр Ar (Арнольд) (требуется в некоторых случаях)

5.1 Построение кода аутентификации (A-кода) (математическая модель)

Пусть S — источник информации, который наблюдается A.

$S = \{s_i \mid i = 1 \ldots |S|\}$ — эти сообщения надо передать с аутентификацией

$M = \{m_i \mid i = 1 \ldots |\mathrm{M}|\}$ —- множество сообщений, полученных в результате кодирования

$E = \{e_i \mid i = 1 \ldots |E|\}$ — множество алгоритмов кодирования с аутентификацией: $\forall i : e_i : S \to M$

На все e_i распространяются такие условия:

1. e_i — инъекция, однако может быть многозначной (в этом случае инъективность понимается в том смысле, что образы различных входов не пересекаются). Будем считать, что $M = \bigcup\limits_{e \in E} e\,(S)$ — объединение $e\,(S)$

2. Существет обратное отображение. Для $\forall e \in S$ определим обратное отображение

$$f_e : M \to S \cup \{0\}$$

То есть

$$\begin{cases} \forall s \in S, e \in E & : f_e\left(e\left(s\right)\right) = S, \\ \forall m \notin e\left(S\right) & : f_e\left(m\right) = 0. \end{cases}$$

Пример 5.1. Если e —- алгоритм шифрования с ключом K, то $f_e =?$ — алгоритм шифрования на том же ключе.

Код аутентификации — тройка $AK =< S, E, M >$, где

- S — множество сообщений (открытых текстов) = состояний источника сообщений
- E — множество кодов $e : S \to M$
- M — множество сообщений для передачи (закодированных сообщений) и выполняются условия 1 и 2 для отображений e

Свойства:

- Множество кодов E — открытая информация и известна аналитику
- A и B предварительно строят A-код и выбирают из множества E случайное отображение для работы — и выбранное отображение является секретом (общим секретом A и B)
- Для передачи информации о состоянии источника S A вычисляет $m = e\left(s\right)$ — сообщение с аутентификацией; m передается B
- Критерий аутентичности сообщения, который применяет B:

  $f_e\left(m\right) \neq 0$ — сообщение подлинно

  $f_e\left(m\right) = 0$ — сообщение не подтверждено

Дополнительные свойства

- $|M| \geq |S|$ (из инъективности)
- если $|M| = |S|$, то

  $$\left(\forall m \in M\right)\left(\forall e \in E\right)\left(\nexists s \in S\right) : e\left(s\right) = m$$

  по сути это приведёт к тому, что все сообщения будут аутентичними и сама схема не пройдет
- $|M| > |S|$ (обычно даже $|M| \gg |S|$)
- Пусть

  $$E\left(m\right) = \{e \mid m \in e\left(S\right)\} = \{e \mid \exists s : e\left(s\right) = m\}$$

  Если $|E(m)| = 1$, то всё, аналитик сразу нашел правильное $e \Rightarrow$ всё взломано. Отсюда новое условие на A-код: $\forall m : |E\left(m\right)| > 1$.

С другой стороны, если $|E(m)| = |E|$, то любое сообщение подойдет как правильное, так как

$$(\forall e)(\exists s) : e(s) = m.$$

Криптоаналитик тогда может найти ложное сообщение из множества $\bigcap\limits_{e \in E} e(S)$ Уточнение к написанному (и новое условие на A-код)

$$\forall m : |E(m)| < |E|$$

- Если $\bigcap\limits_{e \in E(m)} e(S) > 1$, то аналитик всегда сможет подобрать ложные сообщения, которые будут трактоваться как правильные (подделка). Отсюда следует последнее, пятое условие на А-код:

$$\bigcap_{e \in E(m)} e(S) = \{m\}$$

Пример 5.2. Если e — многозначная функция, то A-код называется кодом с расщеплением (если однозначная, то код без расщепления).

Задать A-код можно матрицей $|E| \times |M|$ — матрицей A-кода.

Пусть $S = \{H, T\}$, $M = \{m_1, m_2, m_3, m_4\}$, $E = \{e_1, e_2, e_3, e_4\}$.

Расщепление: $e_3(H) = \{m_2, m_3\}$

e \ m	m_1	m_2	m_3	m_4
e_1	H	T	0	0
e_2	H	0	T	0
e_3	0	H	H	T
e_4	0	0	T	H

Если в каком-то столбике будет только один символ (как в m_1, то это будет код без секрета — аналитик узнает, какое было сообщение, однако оно всё равно будет аутентифицировано.

Проверим выполнение всех условий:

1. Условие
$$M = \bigcup_e (S)$$
Проверим
$$\begin{aligned} e_1(S) &= \{m_1, m_2\} \\ e_2(S) &= \{m_1, m_3\} \\ e_3(S) &= \{m_2, m_3, m_4\} \\ e_4(S) &= \{m_3, m_4\} \end{aligned}$$
выполняется.

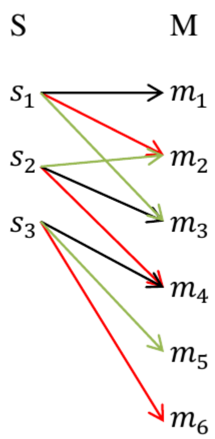
2. $\forall s : f_e(e(s)) = s$ — выполняется по построению.

3. $|M| > |S|$ — выполняется.

4. Условие
$$\forall m : |E| > |E(m)| > 1$$
Проверка
$$\begin{aligned} E(m_1) &= \{e_1, e_2\} \\ E(m_2) &= \{e_1, e_3\} \\ E(m_3) &= \{e_2, e_3, e_4\} \\ E(m_4) &= \{e_3, e_4\} \end{aligned}$$
выполняется.

5. Условие
$$\forall m : \bigcap_{e \in E(m)} e(S) = \{m\}$$
Проверка
$$\begin{aligned} &m_1 : e_1(S) \cap e_2(S) = \{m_1\} \\ &m_2 : e_1(S) \cap e_3(S) = \{m_2\} \\ &m_3 : e_2(S) \cap e_3(S) \cap e_4(S) = \{m_3\} \\ &m_4 : e_3(S) \cap e_4(S) = \{m_3, m_4\} \end{aligned}$$
не выполняется

5.2 A-коды с аутентификатором, т.е. с добавочной информацией для аутентификации (имитовставки, цифровые подписи)



$$E = \{e_1, e_2\} \cup \{e_3\}$$
$$e_1 : \qquad s_1 \to m_1, \qquad s_2 \to m_3, \qquad s_3 \to m_4$$

$$e_2: \qquad s_1 \to m_3, \qquad s_2 \to m_2, \qquad s_3 \to m_5$$

$$e_3: \qquad s_1 \to m_2, \qquad s_2 \to m_4, \qquad s_3 \to m_6$$

Проверяем условия:

1. $$M = \bigcup_e e\left(S\right)$$

   Не выполняется: m_6 не достижимо.

   $$e_1\left(S\right) = \left\{m_1, m_3, m_4\right\}, \qquad e_2\left(S\right) = \left\{m_2, m_3, m_5\right\}$$

   Но с e_3 будет выполняться:

   $$e_3\left(S\right) = \left\{m_2, m_4, m_6\right\}$$

2. $\forall s: f_e\left(e\left(S\right)\right) = S$ — выполняется

3. $|M| > |S|$ — таки да

4. $\forall m: 1 < |E\left(m\right)| < |E|$

   $E\left(m_1\right) = \left\{e_1\right\}$ — очень плохо

   $E\left(m_2\right) = \left\{e_2, e_3\right\}$

   $E\left(m_3\right) = \left\{e_1, e_2\right\}$

   $E\left(m_4\right) = \left\{e_1, e_3\right\}$

   $E\left(m_5\right) = \left\{e_2\right\}$ — очень плохо

   $E\left(m_6\right) = \left\{e_3\right\}$ — очень плохо

   Если аналитик пронаблюдает m_1, m_5, m_6 он будет точно знать, что используется отображение e_1, e_2, или e_3 соответственно (и узнаёт состояние, которое защищалось).

   $\Rightarrow$ Легко сделать обман/подделку (потому что известно кодирующее отображение).

   Если же |E(m)|=|E| и m раньше не посылалось, то аналитик просто отправляет m Бобу, и тот его с радостью принимает, потому что какое бы ни использовалось кодирующее отображение, m будет правильным сообщением.

5. $\forall m: \bigcap\limits_{e \in E(m)} e\left(S\right) = \left\{m\right\}$ не выполняется уже для m_1:

   $$\bigcap_{e \in E(m)} e\left(S\right) = \left\{m_1, m_3, m_4\right\}$$

   Аналитик ловит сообщение m и обнаруживает, что

   $$\bigcap_{e \in E(m)} e\left(S\right) = \left\{m, m'\right\} \Rightarrow m'$$

   тоже будет правильным сообщением, хотя e осталось неизвестным.

! Для того, чтобы аналитик не смог подделать сообщение путём случайного выбора, необходимо, чтобы $\forall c : \{m : f_e(m) = 0\}$ было большим.

В этом случае случайный выбор m попадет на 0 с большой вероятностью.

A-код без секретности — не скрывает состояние S.

В этом случае можно и представить его в виде A-кода с аутентификатором: $M \subseteq S \times A$.

$m = (s, a)$, a — аутентификатор.

Пусть $\overline{e} : S \to A$ — функция вычисления аутентификатора.

5.2.1 Построение A-кода с аутентификатором по A-коду без секретности

Обозначим:

$$\begin{aligned} M(S) &= \{m \in M \mid \exists e \in E : e(S) = m\} \\ \mu_s &= |M(S)| \\ \mu &= \max_s \mu_s \end{aligned}$$

Зная μ, выбираем произвольное множество $A = \{a_1, \ldots, a_\mu\}$, которое будем использовать в качестве аутентификаторов.

Новое множество сообщений $M = \{m_1, \ldots, m_\gamma\} \subseteq S \times A$?

Тогда $M(S) = \{m_{i_1}, \ldots, m_{i_{\mu_s}}\}$, $\mu_{i_j} = e_j(S)$?

Положим $\overline{e_j(S)} = a_j$, $j = \overline{1, \ldots, \mu_s}$ и поставим взаимно-однозначное соответствие: $M(S) \leftrightarrow M_s$

$$M_s = \{(s_1, a_1), (s_1, a_2), \ldots, (s_1, a_{\mu_s})\}$$

Тогда новое множество сообщений $\overline{M} = \bigcup\limits_s M_s$

Пример 5.3.

$$S = \{H, T\}, \qquad M = \{m_1, \ldots, m_5\}, \qquad E = \{e_1, \ldots, e_6\}$$

e \ m	m_1	m_2	m_3	m_4	m_5
e_1	H	T	0	0	0
e_2	0	0	H	T	0
e_3	0	0	H	0	T
e_4	H	0	0	T	0
e_5	0	T	H	0	0
e_6	H	0	0	0	T

Секретности нет: каждое m_i получается либо только из H, либо только из T.

$$\begin{aligned} M(H) &= \{m_1, m_3\} \\ M(T) &= \{m_2, m_4, m_5\} \\ \mu &= 3 \\ \mu_H &= 2 \\ \mu_T &= 3 \end{aligned}$$

$A = \{a_1, a_2, a_3\}$ — аутентификаторы $\{a, b, c\}$ Тогда

$$M_H = \{(H, a), (H, b)\}$$
$$M_T = \{(T, a), (T, b), (T, c)\}$$
$$\overline{M} = M_H \cup M_T \subseteq S \times A$$

Отображение $M \to \overline{M}$:

$$m_1 \to (H, a), \qquad m_2 \to (T, a), \qquad m_3 \to (H, b),$$
$$m_4 \to (T, b), \qquad m_5 \to (T, c).$$

Оглавление