11. Понятие надежности. Методология обоснования надежности криптографической защиты.

Под надёжностью шифра понимают его стойкость, т.е. устойчивость перед атаками криптоаналитиков.

Шеннон вводит два вида стойкости: практическую и теоретическую. При этом он берёт в рассмотрение только атаку по одному шифртексту, т.е. когда противник перехватывает одну единицу шифртекста и больше никаких дополнительных данных.

11.1. Общие обозначения

Для рассмотрения стойкости шифров введём следующие общие обозначения:

- X – множество всех открытых текстов для данного шифра
- Y – множество всех шифртекстов для данного шифра
- K – множество всех ключей для данного шифра
- $H(x)$ – Шенноновская энтропия элементов некоторого множества x

11.2. Теоретическая стойкость

Чтобы дать определение теоретической стойкости шифра, необходимо ввести понятие условной энтропии.

Пусть у нас есть распределения вероятностей $P(X), P(Y)$ для множеств X и Y, т.е. каждому элементу из этих множеств поставлена в соответствие некоторая вероятность его появления.

Тогда условным распределением $P(X/y)$ назовём вероятности того, что каждый элемент $x_i \in X$ после зашифрования дал конкретный шифртекст y. Это распределение будет состоять из вероятностей $p(x_i/y)$, каждая из которых показывает вероятность того, что конкретный открытый текст x_i при зашифровании превратился в шифртекст y_i. Если есть распределение, можем посчитать и энтропию. В случае условного распределения это будет условная энтропия:

$$H(X/y) = -\sum_{x \in X} p(x/y) \cdot log_2 p(x/y)$$

Такая энтропия показывает, насколько мы, зная конкретный шифртекст y, не уверены в том, из какого конкретного открытого текста x_i он был получен.

Чтобы обобщить это на все шифртексты, нужно усреднить эту энтропию по всем возможным шифртекстам. Тогда мы получим условную энтропию двух распределений X и Y:

$$H(X/Y) = -\sum_{y \in Y} \sum_{x \in X} p(y) \cdot p(x/y) \cdot log_2 p(x/y)$$

Теперь рассмотрим, как характеризует шифр такая энтропия.

Пусть, например, $H(X/Y) = 0$. Это будет значить, что знание конкретного шифртекста y не оставляет никакой неопределённости насчёт конкретного открытого текста x, из которого y был получен. Т.е., зная шифртекст, мы можем с точностью определить открытый текст. Очевидно, это плохой шифр.

Шеннон показывает, что $H(X/Y) \leq H(X)$, т.е. максимальное возможное значение для $H(X/Y)$ есть $H(X)$. Рассмотрим, что значит равенство $H(X/Y) = H(X)$. При условии такого равенства, мы можем говорить, что знание шифртекста никак не изменило наши предположения о том, какой открытый текст был зашифрован. Такой шифр явно хорош. Более того, согласно Шеннону, такой шифр называется идеальным и возможен для реализации только в определённых условиях.

Таким образом, вводятся два критерия теоретической стойкости шифра:

1) Неопределённость шифра по открытому тексту – $H(X/Y)$

2) Неопределённость шифра по ключу – $H(K/Y)$

При этом Шеннон показал следующее соотношение:

$$H(K/Y) = H(X) + H(K) - H(Y)$$

Неопределённость шифра по ключу определяет ответ на следующий вопрос: зная конкретный шифртекст, насколько мы не уверены в том, с помощью какого ключа он был получен?

11.3. Практическая стойкость

Чтобы дать определение практической стойкости шифра, следует ввести определение расстояния единственности.

Пусть есть некоторый язык Λ и R_Λ – избыточность этого языка.

$R_\Lambda = 1 - \frac{H_\Lambda}{log_2 n}$, где H_Λ – избыточность языка Λ, n – длина алфавита этого языка.

В свою очередь $H_\Lambda = \lim\limits_{r \to \infty} \frac{H_r}{r}$, где r – длина r-грамм языка, по которым считается энтропия.

Тогда для шифра, шифрующего язык Λ с длиной алфавита n расстояние единственности определяется так:

$$L_0 = \left\lceil \frac{log_2|K|}{R_\Lambda \cdot log_2 n} \right\rceil$$

L_0 – минимальная длина шифртекста, для которого только один ключ при расшифровании даст осмысленный открытый текст. Для всех шифртекстов длиной менее L_0 один шифртекст при расшифровании на разных ключах может давать разные осмысленные шифртексты, таким образом, вводя криптоаналитика в заблуждение насчёт того, какой из подобранных ключей верный. Важно, что вполне возможно равенство $L_0 = \infty$.

Шеннон ввёл понятие рабочей характеристики шифра $W(N)$, показывающей средний объём работы, выраженный в некотрых элементарных вычислительных операциях, который необходим, чтобы определить ключ по шифртексту длиной N.

Значение $W(N)$ растёт до тех пор, пока N не достигнет L_0, после чего уменьшается до некоторого предела.

В качестве значения практической стойкости шифра берётся как раз это предельное значение. Т.е. значение средней работы критпоаналитика при неограниченном объёме шифртекста. Однако на практике вычислить $\lim\limits_{N \to \infty} W(N)$ достаточно сложно, поэтому вводят величину, называемую достигнутой оценкой рабочей характеристики $W_a(\infty) = \lim\limits_{N \to \infty} W_a(N)$ – среднюю трудоёмкость наилучшего из известных методов вскрытия данного шифра.

Оценивать такое значение должен эксперт-криптоаналитик. При этом задаётся какая-то пороговая трудоёмкость Δ, и все шифры, для которых $W_A(\infty) > \Delta$ считаются надёжными.

12. Автоматное определение шифра. Криптографические параметры узлов и блоков шифрующих автоматов.

13. Методы получения псевдослучайных последовательностей.

14. Генераторы псевдослучайных последовательностей и их свойства.

Лекуер предложил следующее обобщённое описание детерминированного ГПСЧ.

ГПСЧ описывается следующим набором параметров: (S, μ, f, U, g).

- S – множество состояний ГПСЧ;
- μ – распределение вероятностей выбора начального состояния s_0;
- f – функция перехода $f : S \to S, s_i = f(s_i - 1)$;
- U – множество выходных значений, как правило, $U = \{0, 1\}$;
- $g : S \to U$.

Периодом ГПСЧ называют число переходов, после которого состояния и, следовательно, генерируемые значения начинают повторяться.

Примеры ГПСЧ:

14.1. Линейный конгруэнтный метод

Линейный конгруэнтный метод – один из самых простых, но популярных. Каждое новое случайное число $X_i = (aX_i - 1 + c)\ mod\ m$, где a, c и m – некоторые константы. Период не может быть больше m, но можно сделать его равным m при определённых условиях. Не используется в криптографии, т.к. значения предсказуемы. На основе этого генератора разрабатывались многие другие, например, полиномиальные конгруэнтные методы, но для них всех также было показано, что значения можно предсказать. Также стоит отметить, что биты генерируемых чисел зачастую не равновероятны, поэтому часто берут только некоторые битовые части получившегося числа. Однако их использование для моделирования вполне оправдано.

14.2. Запаздывающие генераторы Фибоначчи (аддитивные генераторы)

Это целое семейство генераторов, суть которых в том, что каждое следующее число зависит от нескольких уже сгенерированных сколько-то шагов назад. В общем виде аддитивные генераторы работают по следующей формуле: $X_i = (X_{i-a} + X_{i-b} + X_{i-c} + ... + X_{i-m}) \ mod \ 2^n$. Соответственно, в качестве начального состояния необходимо задать массив из m значений. Если хорошо подобрать коэффициенты a, b, c и т.д., можно сделать так, чтобы период генератора был не меньше $2n - 1$. Например, распространённый фибоначчиев генератор работает по такой формуле:

$$X_k = \begin{cases} X_{k-a} - X_{k-b}, & \text{если } X_{k-a} \geq X_{k-b}, \\ X_{k-a} - X_{k-b} + 1, & \text{если } X_{k-a} < X_{k-b}, \end{cases}$$

В данном случае X_i – вещественные числа. Для работы генератору нужно держать в памяти $max(a, b)$ чисел, изначально они могут быть сгенерированы конгруэнтным методом. Период такого генератора может быть оценен как $T = (2^{max(a,b)} - 1) \cdot 2^l$, где l – число битов в мантиссе вещественного числа.

Фибоначчиевы генераторы создают последовательности, обладающие хорошими статистическими свойствами, причём для всех бит.

14.3. Регистры сдвига с линейной обратной связью (LFSR)

LFSR – сдвиговый регистр битовых слов, у которого значение входного (вдвигаемого) бита равно линейной булевой функции от значений остальных битов регистра до сдвига. Используется как ГПСЧ для генерации одиночных бит.

Булева функция, как правило, строится следующим образом. Каждый из L бит регистра умножается на некоторый коэффициент $c_i \in \{0, 1\}, i = \overline{1, L}$, причём $c_L = 1$. Дальше все результаты умножения складываются xor-ом. Таким образом, коэффициенты c_i формируют характеристический многочлен степени L над полем $GF(2)$ для данного регистра. Чтобы период генерации случайных чисел был максимальным $(2^L - 1)$, нужно, чтобы многочлен был примитивным.

Основные недостатки: медленно работает, многочлен можно определить, получив $2L$ бит с генератора.

14.4. Регистры сдвига с обобщённой обратной связью (GFSR)

GFSR – продолжение развития LFSR. Чтобы построить GFSR нужно:

1) Взять LFSR с примитивным характеристическим трёхчленом $x^p + x^{p-q} + 1$, где p и q – некоторые натуральные числа, $q < p$.

2) Сгенерировать неповторяющуюся последовательность максимальной длины $(2^p - 1)$, произвольно задав начальное состояние.

3) Составить таблицу из p строк, в которой каждая строка – сгенерированная на предыдущем шаге последовательность, циклически сдвинутая на некоторое фиксированное кол-во бит вправо.

4) Столбцы этой таблицы и есть значения нового генератора, их тоже $2^p - 1$ штук, но каждое из них уже представляет не бит, а p-битное слово.

14.5. Вихрь Мерсенна

Вихрь Мерсенна – очень популярная вариация GFSR. Используется во многих ЯП, например, в C++. Есть несколько вариаций, но самая распространённая – MT19937 ($2^{19937} - 1$ – период генератора).

Строится он похоже на GFSR:

1) Выстраивается таблица из 624 строк W_i, каждая по 32 бита.

2) Генерируется промежуточная строка tmp, равная первому (старшему) биту первой строки таблицы, конкатенированному с младшими 31 битами второй строки таблицы.

3) Полученная строка tmp объединяется xor-ом с 227-ой строкой таблицы и таким образом получается случайное 32-битное число.

4) Таблица сдвигается вверх удалением первой строки и добавлением полученного случайного числа в качестве последней строки.

Генератор получил такую популярность благодаря, во-первых, огромному периоду $(2^{19937} - 1)$, во-вторых, прекрасным статистическим свойствам.

15. Блочные и поточные шифры.

Блочные шифры преобразуют открытый текст в шифртекст блоками по несколько байт (в зависимости от реализации шифра), поточные, как правило, преобразуют по одному байту или (если верить Шнайеру) биту. Ввиду такой неоднозначности с "маленькой порцией сообщения", преобразуемой поточным шифром за один раз, Шнайер приводит следующий способ различия блочных и поточных шифров: блочный шифр преобразует данные большими блоками с помощью фиксированного преобразования, а поточный применяет изменяющееся во времени преобразование к отдельным символам открытого текста.

Общее требование к блочным шифрам, сформированное Шенноном – принцип "перемешивания" – гласит, что незначительные изменения открытого текста должны приводить к значительным изменениям шифртекста.

Для выполнения этого требования блочные шифры строятся следующим образом. При зашифровании над каждым блоком открытого текста итерационно повторяются два типа преобразований: криптографически сложные преобразования частей блока и простые перестановки частей в пределах блока.

Примеры блочных шифров: AES, ГОСТ Р 34.12-2015.

Поточные шифры реализуются по следующему общему принципу. Есть некоторый генератор потока ключей, который по определённому правилу генерирует биты некоторой гаммы, зависящей от ключа симметричного шифрования. Для зашифрования генерируемая гамма, бит за битом, накладывается на соответствующие биты открытого текста с помощью xor. Для расшифрования та же самая гамма должна быть наложена на биты шифр текста.

Очевидной проблемой в данном подходе является синхронизация. Генераторы ключевых последовательностей должны работать идентично на передающей и приёмной сторонах. К такой синхронизации существуют два общих подхода:

1) Самосинхронизация по шифртексту. В таком случае каждый следующий бит гаммы является функцией от основного ключа и n предыдущих бит шифртекста. Этот подход в некоторой степени устойчив как к потере, так и к искажению отдельного бита. В обоих случаях будут искажены только n бит расшифрованного текста.

2) Синхронные потоковые шифры. Здесь поток ключей генерируется независимо от сообщения. Здесь обе стороны всегда должны иметь

полностью синхронизированные состояния своих генераторов. Для этого, как правило, перед началом работы задаётся какое-то начальное состояние. Т.к. смена состояния зависит только от номера текущего зашифровываемого или расшифровываемого бита, такой подход неустойчив к потере бита. Одна потеря приведёт к неверному расшифрованию всего последующего сообщения. Чтобы избежать этого, используются, например, специальные "маркеры" – некоторые данные вставляемые в сообщение с определённым периодом и показывающие, в каком состоянии должен находиться генератор к моменту расшифрования данного бита. Однако такой подход гораздо более устойчив к искажению бита. Искажение одного бита повлияет только на соответствующий бит полученного расшифрованного текста и ни на какие другие. Если учесть, что потеря бита является гораздо более редкой ситуацией, чем искажение, этот подход не выглядит так уж плохо.

Стоит отметить, что блочные шифры при использовании в некоторых режимах (см. вопрос 16) по сути представляют собой поточные шифры с самосинхронизацией, где в качестве размера символа берётся размер блока.

Поточные шифры чаще всего реализуются с помощью ГПСЧ (см. вопрос 14). Например, чтобы реализовать генератор ключевой последовательности с помощью LFSR, достаточно взять несколько LFSR с разными характеристическими многочленами и длиной, а начальное состояние задать с помощью ключа. Дальше для генерации нового ключевого бита используется сдвиг всех регистров, а ключевой бит определяется как функция некоторых бит из этих регистров. При этом LFSR могут быть по-разному связаны между собой и влиять друг на друга. Например, Шнайер приводит несколько генераторов вида Stop-and-Go, в которых выход одного LFSR определяет, будет ли осуществляться сдвиг одного или нескольких других LFSR или нет.

16. Режимы использования блочных шифров.

Блочные шифры зашифровывают представленную информацию блоками фиксированной длины. Однако, если применять такие шифры непосредственно (что соответствует первому режиму далее), т.е. независимо шифровать каждый отдельный блок, шифр будет в какой-то степени уязвим к атаке "со словарём", т.к. блоки фиксированной длины могут повторяться в открытом тексте, и, набрав достаточное количество открытых и

зашифрованных текстов, злоумышленник простым сопоставлением сможет раскрывать некоторые блоки. Чтобы такого не происходило, вводятся более сложные способы (режимы) использования блочных шифров.

Согласно ГОСТ Р 34.13-2015 “Режимы работы блочных шифров”, существуют следующие режимы:

1) Режим простой замены (Electronic Codebook, ECB)
2) Режим гаммирования (Counter, CTR)
3) Режим гаммирования с обратной связью по выходу (Output Feedback, OFB)
4) Режим простой замены с зацеплением (Cipher Block Chaining, CBC)
5) Режим гаммирования с обратной связью по шифртексту (Cipher Feedback, CFB)
6) Режим выработки имитовставки (Message Authentication Code algorithm)

Для шифрования, очевидно, используются только первые 5 из них. Последний предназначен для выработки имитовставки (см. вопрос 17). При этом первый не рекомендуется к использованию по причинам, описанным в начале данного вопроса.

Если открытый текст по длине не кратен длине блока (или другому размеру, необходимому для работы соответствующего режима), он дополняется по одному из трёх алгоритмов, приведённых в том же ГОСТе.

Во всех режимах, кроме первого, используется синхропосылка IV (initialization vector, я полагаю) – двоичный вектор определённой длины. Она задаёт некоторые начальные значения для работы режимов и должна быть известна обеим сторонам для каждого шифруемого сообщения. При этом к ней предъявляются следующие требования:

Для режимов 4 и 5 синхропосылка должна выбираться случайным образом равновероятно из всех возможных двоичных векторов.

Для режима 2 синхропосылка должна быть разной для всех сообщений, зашифрованных на одном и том же ключе.

Для режима 3 значение синхропосылки должно быть либо случайным либо уникальным.

Режим простой замены работает тривиально. Для зашифрования применяем блочный алгоритм к каждому блоку открытого текста, для расшифрования – к шифртексту.

Для режима гаммирования сначала вырабатывается гамма – битовый вектор, равный длине открытого текста. Вырабатывается он с помощью

шифрования блочным шифром значений счётчика, причём начальное значение задаётся синхропосылкой. Потом гамма накладывается на открытый текст с помощью xor. Расшифрование, соответственно, точно так же.

Для гаммирования с обратной связью по выходу используется регистр сдвига. Изначально он заполняется синхропосылкой. Для выработки гаммы левая часть регистра зашифровывается блочным шифром, из чего получается гамма для очередного блока открытого текста. Потом содержимое регистра сдвигается на длину этой гаммы, а сама гамма записывается в регистр справа и процедура выработки повторяется для следующего блока. Гамма на блоки так же накладывается и "снимается" по xor.

Простая замена с зацеплением. Почти то же самое. Есть регистр сдвига длина больше, чем блок шифра. Сначала он заполняется синхропосылкой. Потом его старшая часть накладывается на открытый текст хором, и результат зашифровывается. Регистр сдвигается влево на размер блока, а его младшая часть заполняется полученным зашифрованным блоком. Процедура повторяется.

То же самое, что и гаммирование с обратной связью по выходу, только в регистр справа дописывается не полученная на предыдущем шаге гамма, а результат её наложения на открытый текст, т.е. очередной блок шифртекста.

При выработке имитовставки в результате получается двоичный вектор фиксированной длины, в отличие от зашифрования. Для выработки имитовставки сначала из основного ключа генерируются два вспомогательных ключа, каждый по длине равен длине блока алгоритма шифрования. Для этого сначала на основном ключе зашифровывается нулевой вектор длиной как блок, потом с помощью сдвигов и xor с константой вырабатываются остальные. Дальше имитовставка вырабатывается просто: очередной блок зашифровывается блочным алгоритмом с основным ключом $\rightarrow$ результат зашифрования накладывается по xor на следующий блок открытого текста $\rightarrow$ полученный блок снова зашифровывается блочным алгоритмом с основным ключом. Исключением является последний блок. Для его вычисления берётся xor последнего блока открытого текста, предыдущего полученного блока имитовставки и один из вспомогательных ключей: первый, если блок полный, второй, если нет (блок дополняется). Результат этого xor также зашифровывается блочным алгоритмом на основном ключе.

17. Алгоритмы выработки имитовставки. Методы оценки имитозащищенности.

Имитовставка (MAC, англ. message authentication code) – средство обеспечения защиты от навязывания ложных данных в протоколах аутентификации сообщений с доверяющими друг другу участниками – специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности и аутентификации источника данных.

Проблема. Обычное средство обеспечения целостности – хеш функция. Передающая сторона вычисляет хеш от своего сообщения и прикрепляет к сообщению. Принимающая сторона тоже вычисляет хеш от сообщения и сверяет с прикреплённым. Если получено равенство, сообщение с высокой вероятностью неизменно. Однако при такой схеме злоумышленник тоже может посылать принимающей стороне сообщения, хешируя их тем же алгоритмом, и принимающая сторона никак не отличит подделку.

Для решения проблемы в вычисление хеш-функции каким-либо образом вводится секретный ключ, известный только легитимным сторонам взаимодействия: отправителю и получателю. Самый простой способ – просто зашифровать хеш сообщения каким-то симметричным алгоритмом. Есть способы сложнее, например, специальный режим выработки имитовставки в ГОСТ Р 34.13-2015 (см. вопрос 16). Также есть способ CBC-MAC, где в качестве значения имитовставки берётся последний блок сообщения зашифрованного блочным алгоритмом в режимах CBC или CFB.

У всех этих способов и у имитовставки в целом есть один большой недостаток. Они работают только в случае, если стороны обмена доверяют друг другу. Происходит это по двум причинам:

1) Передающая сторона может отправить какое-то «плохое» сообщение, сгенерировав для него имитовставку, но потом утверждать, что получатель сам сгенерировал это сообщение, т.к. у него тоже есть ключ, и он тоже мог сгенерировать имитовставку.

2) Принимающая сторона может сгенерировать какое-то «плохое» сообщение, сгенерировать имитовставку и утверждать, что получила его от передающей стороны, т.к. у передающей стороны тоже есть ключ и она могла сгенерировать имитовставку.

В каждом из этих случаев в случае использования имитовставки отсутствует возможность проверить, кто из сторон говорит правду. Эта

проблема решается с помощью электронной подписи.

18. Режимы аутентифицированного шифрования. Современные стандартизированные решения.

Аутентифицированное шифрование позволяет обеспечить не только конфиденциальность данных, но и их аутентичность.

Есть два варианта: использовать одну из комбинаций имитовставки и шифрования, либо специальный режим шифрования.

18.1. Имитовставка + шифрование

Комбинировать имитовставку и шифрование можно одним из следующих трёх способов:

1) Encrypt-and-MAC: открытый текст отдельно зашифровывается и для него отдельно вырабатывается имитовставка, т.е. сторона передаёт сообщение вида $E(m), MAC(m)$, где E – зашифрование сообщения, MAC – получение имитовставки для сообщения, m – открытый текст.

2) MAC-then-encrypt: сначала вычисляется имитовставка для открытого текста, присоединяется к открытому тексту (конкатенацией), а потом полученное сообщение зашифровывается, т.е. сторона передаёт сообщение вида $E(m||MAC(m))$, где $||$ – конкатенация битовых строк.

3) Encrypt-then-MAC: сообщение сначала зашифровывается, потом для шифртекста вырабатывается имитовставка и передаётся вместе с шифртекстом, т.е. сторона передаёт сообщение вида $E(m), MAC(E(m))$.

Во всех трёх вариациях необходимо обязательно использовать разные ключи для имитовставки и зашифрования. В целом можно использовать любые алгоритмы выработки имитовставки и шифрования, если они сами по себе достаточно безопасны.

Encrypt-and-MAC является наименее безопасным, т.к. имитовставка разрабатывается в первую очередь для того, чтобы её нельзя было подделать. Т.е. внимание тому, не несёт ли она какой-то информации о тексте,

из которого вырабатывалась, не уделяется, т.к. она чаще всего отправляется вместе с открытым текстом. Таким образом, теоретически имитовставка может дать злоумышленнику дополнительную информацию для криптоанализа передаваемой информации. Однако этот вариант используется в SSH.

MAC-then-encrypt более безопасен, потому что в нём «закрыта» проблема, описанная для предыдущего варианта. Однако, чтобы понять, что сообщение повреждено, принимающей стороне необходимо сначала расшифровать повреждённых шифр текст, что занимает время, а также представляет собой уязвимость для атаки с выбранным шифртекстом (см. вопрос 24). Тем не менее, он использовался в TLS во всех версиях, кроме последней.

Encrypt-then-MAC наиболее безопасен и удобен из этих трёх подходов, потому что принимающей стороне нет необходимости расшифровывать всё сообщение, чтобы понять, что оно повреждено, достаточно проверить имитовставку. Этот вариант используется в IPSec.

18.2. Аутентифицированные шифры

Аутентифицированные шифры – то же, что обычные шифры, но в результате своей работы дают не только шифртекст, но ещё и имитовставку. Т.е. получается следующая схема: $AE(m) = (C, T)$, где C – шифртекст, T – имитовставка, AE – функция аутентифицированного зашифрования (authenticated encryption).

Соответственно, существует функция аутентифицированного расшифрования – AD, $AD(C, T) = m$, которая возвращает ошибку в случае, если имитовставка оказалась неверной.

В современных стандартах, например, TLS 1.3, требуется не просто аутентифицированное шифрование (AE), а аутентифицированное шифрование с ассоциированными данными (AEAD). Это шифрование с аутентификацией, которое позволяет оставлять часть данных незашифрованными, но аутентифицированными. Т.е. MAC применяется ко всем, а зашифрование только к части передаваемых данных. Это нужно, например, при передаче сетевых пакетов, когда сами передаваемые данные должны быть зашифрованы и аутентифицированы, но заголовок, содержащий, например, IP-адрес назначения, должен быть аутентифицирован, но передаваться открыто. Таким образом, вводится функция зашифрования:

$AEAD(m, m_a) = (C, m_a, T)$, где m_a – дополнительные данные, которые не зашифровываются. И соответствующая ей функция расшифрования:

$ADAD(C, m_a, T) = (m, m_a)$.

Дополнительно, чтобы одни и те же открытые тексты не превращались всегда в одни и те же шифртексты, используется вектор инициализации (см. вопрос 16).

Наиболее распространённым из таких протоколов является AES-GCM. На самом деле GCM-mode может использоваться с любым симметричным шифром, но чаще всего встречается именно в связке с AES. GCM это Galois/counter mode. Называется так, потому что использует режим гаммирования (CTR – counter) для зашифрования и умножение в поле Галуа для имитовставки.

Принцип работы проще показать на схеме (Рис. 1), которая уехала в другой вопрос.

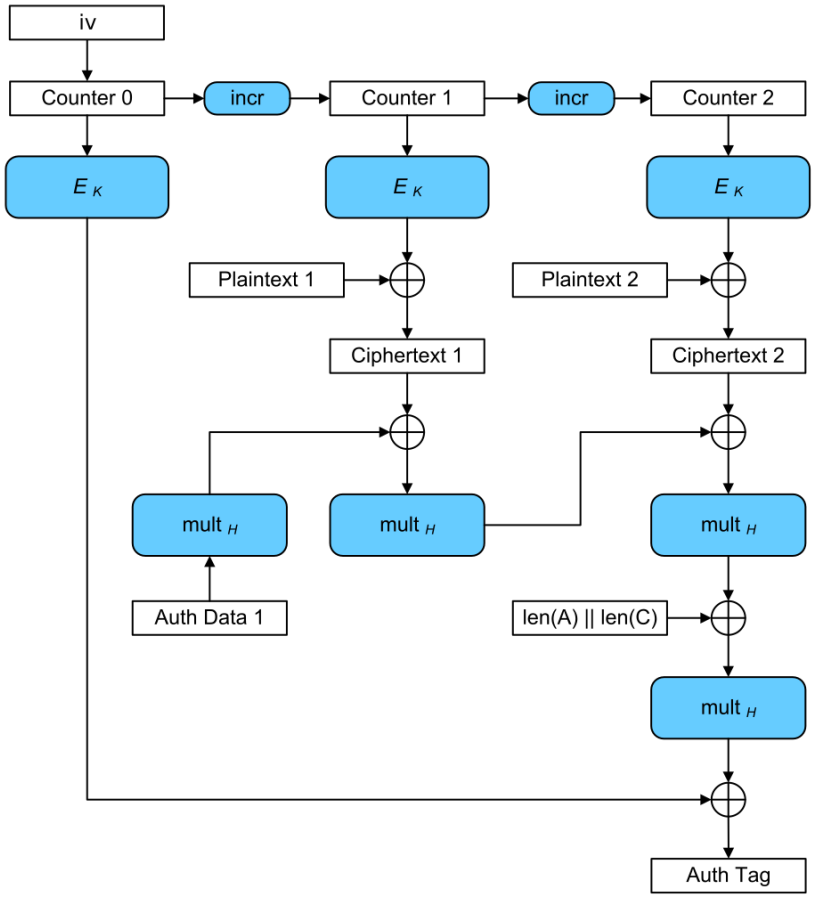

Рис. 1. AES-GCM, к вопросу 18

На рисунке Auth data – открытые данные, которые необходимо присоединить, Plaintex – текст, который всё-таки нужно зашифровать, $mult_H$

– операция умножения полиномов в поле Галуа, $H = E_K(0^{128})$.

Проблемой этого алгоритма является только то, что нельзя использовать повторяющиеся инициализирующие значения.

Помимо AES-GCM существуют и другие аутентифицированные шифры, например, OCB, SIV.

19. Ключевые системы, методы распределения ключей.

–

20. Методы выработки производных ключей, принципы оценки качества производной ключевой информации.

В России выработка производной ключевой информации регламентируется Рекомендациями по стандартизации Р 1323565.1.022–2018.

При реализации средствами криптографической защиты информации (СКЗИ) нескольких криптографических функций возникает необходимость использования для механизмов, реализующих каждую из функций, различных ключей, выработанных из исходной ключевой информации. Исходной ключевой информацией может являться, например, заранее распределенный ключ или ключ, полученный в результате выполнения протоколов выработки общего ключа.

Функции выработки производных ключей осуществляют криптографическое преобразование исходной ключевой информации с использованием дополнительных открытых данных с целью получения ключей для дальнейшего использования в различных функциях.

В описании используются следующие обозначения:

- S – исходная ключевая информация;
- C – представление числа, используемого в итеративных конструкциях в качестве счетчика. Способ представления счетчика должен быть согласован между участниками информационного обмена;
- P – метка использования – двоичная строка, содержащая информацию об использовании вырабатываемых производных ключей. Может содержать, например, информацию о конкретном механизме,

для которого предназначается производный ключ (ключ шифрования ключей, ключ шифрования данных, ключ имитозащиты и т. п.) или, в случае одновременной выработки ключей для нескольких примитивов, информацию о разделении производного ключевого материала на различные производные ключи; допустимые значения и способ представления должны быть согласованы между участниками информационного обмена;

- U – информация об участниках информационного обмена, которыми предполагается ис пользование вырабатываемой ключевой информации;

- может включать в себя идентификаторы пользователей и прочую информацию, известную всем участникам, вырабатывающим производную ключевую информацию;

- A – некоторая дополнительная информация, используемая при выработке производной ключевой информации, например, метка времени;

- L – длина (в двоичной записи) вырабатываемого производного ключевого материала в битах;

- T – соль – случайная строка фиксированной длины, обычно вырабатываемая в момент выполнения алгоритма.

Функции выработки производных ключей принимают на вход шесть аргументов: исходную ключевую информацию S, длину производной ключевой информации L, соль T, метку использования T, информацию о субъектах U, дополнительную информацию A.

Функции состоят из двух этапов. На первом этапе из исходной ключевой информации и соли вырабатывается промежуточный ключ длины 256 бит. Полученный промежуточный ключ вместе с остальными входными параметрами используется на втором этапе функций, выходом которых является производный ключевой материал K_0 длины L. Первый этап функции обозначается $kdf^{(1)}$ и возвращает промежуточный ключ $K^{(1)}$, второй обозначается $kdf^{(2)}$ и возвращает производный ключ K_0.

В качестве промежуточных преобразований используют ключевые функции хеширования $HMAC_K^{(n)}$ и $NMAC_K^{(256)}$, вырабатывающие имитовставку длины n и 256 бит соответственно. Обе эти функции в качестве базовой хеш-функции ($Hash$) используют хеш-функцию из ГОСТ Р 34.11-2012:

$$HMAC_K^{(n)}(X) = Hash^{(n)}\left((K \oplus C_{OUT})||Hash^{(n)}((K \oplus C_{IN})||X)\right)$$

$$NMAC_K^{(256)}(X) = Hash^{(256)}\left((K \oplus C_{OUT})||Hash^{(512)}((K \oplus C_{IN})||X)\right)$$

Здесь C_{IN} и C_{OUT} – константы, заданные в стандарте, $||$ – операция конкатенации битовых строк.

В качестве функции выработки промежуточного ключа используется одна из следующих:

1) $K^{(1)} = NMAC_T^{(256)}(S),$ при этом $|T| \leq 512$

2) $K^{(1)} = LB_{256}(HMAC_T^{(512)}(S)),$ при этом $|T| \leq 512$

3) $K^{(1)} = S \oplus T,$ при этом $|T| = 256$

Здесь $LB_x(X)$ – взятие x старших (левых) бит строки X, $|X|$ – длина строки X.

Далее для выработки результирующего производного ключа используют следующий алгоритм:

1) $C_0 = 0$

2) $z_0 = IV$

3) Для $i = \overline{1, \lceil L/n \rceil}$ выполнять:

   3.1) $C_i = C_{i-1} + 1$

   3.2) $z_i = MAC_{K^{(1)}}^{(n)}(format(z_i - 1, C_i, P, U, A, L))$

4) $K_0 = LB_L\left(z_1||z_2||...||Z_{\lceil L/n \rceil}\right)$

Здесь:

- MAC – ключевая функция хеширования, в качестве неё можно взять $HMAC$, $NMAC$ или режим выработки имитовставки из ГОСТ Р 34.13-2015 (см. вопрос 16)
- IV – вектор инициализации – общедоступное значение, которое открыто передаётся перед стартом протокола

- $format$ – общеизвестный способ превратить 6 значений в строку бит (оговаривается сторонами заранее)

Основное требование к производной ключевой информации, применяемой для шифрования и имитозащиты передаваемых сообщений, заключается в невозможности её определения нарушителем с трудоёмкостью, меньшей чем тотальное опробование всех возможных значений. Каждый ключ, как правило, представляется в виде двоичного вектора длины m бит, таким образом, нарушителю необходимо опробовать 2^m ключей для компрометации сообщений. Отсюда следует, что необходимо проверить выполнимость следующих условий:

- Множество значений, которые может принимать производный ключ, совпадает с множеством V_m всех возможных двоичных векторов длины m;

- Принимаемые производным ключом значения непредсказуемы, т. е. последовательность нескольких выработанных в различных сессиях протокола производных ключей $K1, K2, ...$ должна быть статистически неотличима от последовательности случайных равновероятно распределённых на множестве V_m величин.

При практическом применении средств защиты информации могут нарушаться правила эксплуатации средств, превышаться заданные ограничения на объём обрабатываемой информации или возникать уязвимости в программном обеспечении, все вместе или по отдельности приводящие к возможности практического определения нарушителем производных ключей или исходной ключевой информации. Это приводит к необходимости встраивания в криптографические протоколы мер, минимизирующих объём скомпрометированной информации. В качестве таких мер могут выступать:

1) использование односторонних функций, не позволяющих вычислять значения ключей аутентификации по значениям производных ключей;

2) использование в каждой сессии протокола уникальных случайных значений для выработки производных ключей;

3) использование «древовидных» структур выработки производных ключей, не позволяющих нарушителю по известному производному ключу K_n вычислить значения ключей K_{n-1} и K_{n+1}.

Дополнительно в рамках математических исследований должны быть проверены следующие гипотезы:

- о ничтожной вероятности совпадения различных производных ключей, вырабатываемых в рамках одной сессии протокола;
- о статистической независимости последовательности производных ключей $K_1, K_2, ...$, вырабатываемых в различных сессиях протокола.

21. Асимметричные криптографические схемы.

Асимметричные криптографические схемы – протоколы криптографии с открытым ключом.

Основная суть состоит в следующем: каждая сторона информационного обмена имеет пару ключей: открытый и закрытый. При этом из закрытого можно легко вычислить открытый, а вот наоборот – вычислительно сложно.

Открытый ключ публикуется для всех, в т.ч. его может знать злоумышленник. Это нормально, потому что открытый ключ может использоваться только для зашифрования сообщений. Расшифровать такое сообщение можно только с помощью закрытого ключа, который каждая из сторон держит в секрете.

Преимущество в том, что за всё время информационного обмена ни разу не возникает необходимости куда-либо передавать закрытый ключ. Сторона генерирует его один раз и всё время держит у себя.

Чтобы передать сообщение, передающая сторона сначала запрашивает открытый ключ принимающей стороны и потом зашифровывает своё сообщение на этом ключе. Таким образом, только принимающая сторона сможет расшифровать это сообщение.

Такая система уязвима к некоторым атакам, т.к. открытый ключ общеизвестен и злоумышленник может нашифровать себе сколько угодно сообщений и потом просто сопоставлять передаваемый шифртекст со своим набором шифртекстов.

Другая проблема состоит в том, что передающей стороне зачастую необходимо убедиться, что тот ключ, который пришёл её от принимающей стороны, действительно является открытым ключом принимающей стороны, а не «человека посередине», вторгшегося в канал связи. Для этого используется инфраструктура открытых ключей (см. вопрос 23).

Ряд алгоритмов с открытым ключом пригодны также для создания цифровой подписи (см. вопрос 23).

Если переходить к конкретным алгоритмам, из распространённых можно назвать RSA, ElGamal, Rabin и, конечно же, ГОСТ Р 34.10-2012 (который используется только для создания подписи).

Сложность расшифрования шифртекста без знания закрытого ключа, как правило, базируется на сложности некоторой математической проблемы. При этом, зная решение этой проблемы (закрытый ключ), расшифровать информацию достаточно просто. Так, шифр RSA базируется на сложности задачи факторизации больших чисел – разложения этих чисел на простые множители. ElGamal, в свою очередь, основана на сложности вычисления дискретных логарифмов в конечном поле. Rabin – на сложности поиска квадратных корней в кольце остатков по модулю составного числа. ГОСТ – на сложности вычисления дискретного логарифма в группе точек эллиптической кривой.

Многие существующие схемы криптографии с открытым ключом сейчас имеют аналоги с использованием эллиптических кривых.

22. Гибридные схемы шифрования. Практические примеры реализации гибридных схем.

Гибридные схемы шифрования – протоколы, в которых используются симметричные и асимметричные алгоритмы. Сценарий использования, как правило, один: асимметричный алгоритм используется для получения двумя сторонами общего ключа симметричного шифрования, и далее этот ключ используется для шифрования всей остальной информации. Такой ключ симметричного шифрования, как правило, называется сеансовым.

Причины возникновения таких протоколов, в основном, в том, что:

1) Ключи симметричного шифрования необходимо периодически менять, т.к., во-первых, они могут быть скомпрометированы, во-вторых, если злоумышленник накопит достаточно много единиц шифртекста, зашифрованных на одном ключе, это открывает возможности для некоторых атак.

2) Использовать асимметричное шифрование для защиты основных данных неудобно, т.к. асимметричные алгоритмы достаточно медленны в работе (по утверждениям Шнайера, примерно в 1000 раз медленнее симметричных), а также уязвимы к атакам с выбранным открытым текстом.

Таким образом, есть общая наиболее распространённая схема взаимодействия:

1) Принимающая сторона посылает передающей свой открытый ключ.

2) Передающая сторона зашифровывает свой сеансовый ключ симметричного шифрования открытым ключом принимающей стороны и отправляет.

3) Принимающая сторона расшифровывает сеансовый ключ своим закрытым ключом.

4) Таким образом, обе стороны имеют общий ключ симметричного шифрования – сеансовый – и все дальнейшие передаваемые данные в пределах этого сеанса связи шифруются им.

Из практических примеров реализации таких схем можно привести, например, протокол TLS (см. вопрос 31). Так, в версии 1.2 этого протокола в качестве одного из вариантов используется алгоритм RSA для обмена ключами и алгоритм AES в режиме GCM для шифрования основного трафика. Однако чаще в TLS (особенно в версии 1.3 используется всё же алгоритм Диффи-Хеллмана, где каждая сторона дополнительно подписывает свою часть электронной подписью).

Другими примерами являются протоколы SSH, OpenPGP и PKCS#7.

23. Электронная подпись, инфраструктура открытых ключей. Удостоверяющие центры. Методы обеспечения подлинности физических лиц.

В конце ответа на вопрос 17 была поставлена проблема, связанная с тем, что, используя имитовставку, стороны обмена, не доверяющие друг другу, могут приходить к неразрешимым спорам в результате злонамеренных действий одной из сторон. Ключевым моментом в данной проблеме было то, что, т.к. обе стороны информационного обмена владеют общим секретным ключом, они могут сгенерировать имтовставку для любого сообщения, при этом сообщение никак не привязывается к конкретному выработавшему её пользователю. Таким образом, возникает необходимость в алгоритме, который позволил бы производить похожие действия, но так, чтобы можно было уникально идентифицировать

пользователя, создавшего некоторую последовательность для контроля целостности, т.е. необходимо обеспечить не только целостность и аутентичность информации, но и неотказуемость стороны информационного обмена от авторства.

Для этого используется криптография с открытым ключом. У каждого из пользователей есть ключевая пара – открытый и закрытый ключи, при этом открытый известен всем, закрытый – только самому пользователю. Примечательно, что в большинстве алгоритмов с открытым ключом можно использовать две вариации: открытый ключ для зашифрования и закрытый для расшифрования (классический вариант для обеспечения конфиденциальности), и наоборот – закрытый для зашифрования и открытый для расшифрования (то, что нам нужно сейчас).

Таким образом, чтобы обеспечить все необходимые свойства, можно использовать следующий алгоритм. Отправитель может зашифровать отправляемое сообщение своим закрытым ключом и передавать полученный шифртекст вместе с самим сообщением. Тогда получатель, зная открытый ключ отправителя, может его расшифровать и сравнить с сообщением. Соответственно, если получателю удалось расшифровать сообщение открытым ключом отправителя, и сравнение прошло успешно, получатель может быть уверен во-первых, в том, что информация не была подменена, во-вторых, в том, что информацию отправил именно тот отправитель, открытый ключ которого использовался для расшифрования. Более того, получатель теперь не сможет сам сформировать сообщение и выдавать его за сообщение отправителя, т.к. не знает закрытого ключа. Отправитель, в свою очередь, не сможет сформировать сообщение, а потом отказаться от него, т.к. он зашифровал его своим закрытым ключом и каждый, кто знает его открытый ключ, может проверить, что это сделал именно он.

Описанный алгоритм работает достаточно медленно, т.к. сообщения могут быть длинными и приходится зашифровывать всё. Для этого зашифровывается не само сообщение, а его хеш. Полученный элемент данных фиксированной длины называется электронной подписью данного сообщения. Тогда получение такого элемента данных называется формированием подписи (подписанием), а расшифрование и сравнение хеша с хешем сообщения – проверкой подписи.

Однако остаётся ещё одна проблема. Чтобы все знали открытый ключ некоторого пользователя и могли проверять его подпись, нужно, чтобы этот ключ хранился где-то в открытом доступе. Однако при доступе к этому хранилищу в сеанс связи может вторгнуться злоумышленник (человек посередине) и подменить ключ некоторого пользователя своим. Тогда злоумышленник сможет подписывать произвольную информацию

от имени легитимного пользователя.

Чтобы этого избежать, организуется инфраструктура открытого ключа (Public Key Infrastructure – PKI) – система, с помощью которой можно определить, кому принадлежит тот или иной открытый ключ. Ключевыми сущностями в такой системе являются сертификаты и центры сертификации.

Цифровой сертификат – способ сопоставления открытого ключа физическому лицу или уполномоченному агенту, лежащий в основе инфраструктуры открытого ключа.

Центр сертификации (центр сертификатов, удостоверяющий центр) – это центральный орган, служащий посредником между пользователями и удостоверяющий аутентичность их открытых ключей.

Центр сертификации представляет собой организацию, которой по той или иной причине доверяют все стороны информационного обмена. Центр сертификации имеет свою ключевую пару для электронной подписи. Он выдаёт сторонам информационного обмена цифровые сертификаты – электронные документы, содержащие записи вида

$$\{\text{Уникальный ID пользователя}, \text{Открытый ключ пользователя}\}$$

, подписанные подписью центра сертификации (полное содержимое сертификата регламентируется стандартом X.509).

Таким образом, чтобы удостовериться в подлинности ключа стороны А, сторона Б может обратиться к центру сертификации и запросить у него сертификат ключа стороны А. Также каждая из сторон может попросить центр сертификации выдать ей сертификат на какой-то новый ключ.

Проблема, которая остаётся: физически невозможно организовать один центр сертификации на весь мир, значит, нужно несколько. Для этого центры сертификации (ЦС) организуются в иерархическую структуру (дерево), где есть корневой ЦС, котрому доверяют все (например, в России это Минцифры), и который выдаёт сертификаты другим подчинённым ЦС. Те, в свою очередь, могут выдавать сертификаты другим ЦС или конкретным пользователям и т.д.

При такой иерархической структуре, чтобы убедиться в подлинности ключа стороны А, сторона Б должна проверить сертификат, выданный стороне А его ЦС, потом сертификат этого ЦС от другого ЦС и т.д., пока не дойдёт до ЦС, которому доверяет сторона Б. Чтобы сократить этот путь по дереву, существует перекрёстная сертификация, когда узлы дерева выдают сертификаты друг другу.

24. Атаки на криптографические алгоритмы: алгоритмические, алгебраические, статистические.

Нужно рассмотреть следующие атаки:

1) Человек посередине
2) Вскрытие со словарём
3) Атака с выбранным шифртекстом
4) Атака с выбранным открытым текстом

25. Свойства безопасности криптографических протоколов.

Здесь будет прямой копипаст из работы А. Ю. Нестеренко, А. М. Семенов – Методика оценки безопасности криптографических протоколов.

Выделяют следующие свойства безопасности криптографических протоколов:

1) Свойство аутентификации субъекта (участника протокола) другим субъектом (участником протокола) заключается в подтверждении одним субъектом подлинности другого субъекта, а также в получении гарантии того, что субъект, подлинность которого подтверждается, действительно принимает участиев выполнении текущей сессии протокола.Свойство аутентификации субъекта может быть как односторонним, так и взаимным. В последнем случае свойство должно выполняться для всех участвующих во взаимодействии субъектов.
2) Свойство аутентификации сообщения заключается в подтверждении подлинности источника сообщения и целостности передаваемого сообщения. Подлинность источника сообщения означает, что протокол должен обеспечивать гарантии того, что полученное сообщение или его часть были созданы участником взаимодействия в ходе выполнения текущей сессии протоколав некоторый момент времени, предшествующий получению сообщения. Фактически в

рамках данного свойства сообщение однозначно связывается со своими сточником (субъектом, отправившим сообщение), а выполнение свойства гарантирует, что сообщение не было искажено, в частности подделано нарушителем, при передаче по каналам связи.

3) Свойство целостности сообщений заключается в том, что получатель сообщения обладает возможностью проверить, что полученные им данные (или их часть) не были модифицированы, уничтожены и являются теми же самыми данными, что послал отправитель.

4) Свойство защиты от повторов заключается в том, что один раз корректно принятое участником протокола сообщение не должны быть принято повторно. В зависимости от протокола данное свойство может быть сформулировано в виде одного из следующих требований:

   - должна быть обеспечена гарантия того, что сообщение выработано в рамках текущей сессии протокола;
   - должна быть обеспечена гарантия того, что сообщение выработано в рамках заданного интервала времени;
   - сообщение не было принято ранее.

   В отечественной литературе данное свойство часто называют свойством невозможности навязывания ложных сообщений, подразумевая под этим защиту как от повторного принятия истинных сообщений, так и от подделанных нарушителем сообщений (свойство С 2).

5) Свойство неявной аутентификации получателя заключается в том, что протокол должен обладать средствами, гарантирующими, что отправленное сообщение может быть прочитано только теми участниками, для которых оно предназначено. Только законные авторизованные участники должны иметь доступ к данной информации, многоадресным сообщениям или групповому взаимодействию.

6) Свойство групповой аутентификации заключается в том, что законные авторизованные члены заранее определённой группы пользователей могут аутентифицировать источник и содержание информации или группового сообщения. Сюда также входят протоколы, в которых участники группового взаимодействия не доверяют друг другу.

7) Свойство аутентификации субъекта (участника протокола) доверенной третьей стороной. В протоколах, явно реализующих взаимодействие участников с доверенной третьей стороной, данное свойство эквивалентно первому из перечисленных свойств. В случае использования инфраструктуры открытых ключей данное свойство может выполняться косвенно, путём заверения открытых ключей участников взаимодействия электронной подписью удостоверяющего (доверенного) центра; при этом привязка аутентификации субъекта к какой-либо сессии протокола не может быть обеспечена.

8) Свойство конфиденциальности ключа предполагает, что в ходе информационного взаимодействия значение ключа не может стать известным нарушителю, а также легитимным пользователям информационной системы, для которых данный ключ не предназначен. Данное свойство может применяться как к исходной ключевой информации, так и к производным сессионным ключам.

9) Свойство аутентификации ключа предполагает, что один из участников взаимодействия получает подтверждение того, что никакой другой участник, кроме заранее определённого второго участника и, возможно, доверенного центра, не может обладать секретным ключом, выработанным в ходе выполнения протокола.

10) Свойство подтверждения ключа заключается в том, что один из участников взаимодействия получает подтверждение того, что второй участник (или группа участников) действительно обладает заданным секретным ключом и/или имеет доступ к информации, необходимой для выработки заданного секретного ключа.

11) Свойство стойкости при компрометации производных ключей состоит в том, что компрометация производных ключей, т. е. ключей, используемых непосредственно для шифрования и имитозащиты передаваемой информации, не приводит к нарушению других свойств безопасности как в рамках текущей, так и в других сессиях протокола, в частности к компрометации производных ключей, выработанных ранее или планируемых к выработке в дальнейшем. В литературе данное свойство часто называют защитой от «чтения вперед/назад» или используют термин «perfect forward secrecy».

12) Свойство стойкости при компрометации ключа аутентификации состоит в том, что компрометация долговременного ключа аутентификации не приводит к нарушению конфиденциальности информации, переданной до момента компрометации ключа, а в случае

пассивного нарушителя – и к нарушению конфиденциальности информации, передаваемой после завершения текущей сессии протокола. В литературе данное свойство иногда называют защитой от «чтения назад».

13) Свойство формирования новых ключей заключается в том, что протокол, обладающий данным свойством, позволяет формировать уникальные сессионные и/или производные ключи для каждой сессии протокола.

14) Свойство защиты от навязывания ключевых значений гарантирует, что ни один из участников протокола не может навязать значение общего секретного, сессионного или производного ключа по своему выбору другому участнику протокола.

15) Свойство защиты от навязывания параметров безопасности гарантирует, что используемые в ходе выполнения протокола или согласуемые на этапе установления соединения параметры безопасности не могут быть навязаны нарушителем. В качестве параметров безопасности могут выступать наборы используемых криптографических преобразований, численные параметры алгоритмов и алгебраических структур, в которых выполняется протокол, случайные значения, вырабатываемые в ходе выполнения протокола и т. п.

16) Свойство конфиденциальности заключается в том, что данные, передаваемые в ходе информационного взаимодействия, не могут стать известными нарушителю и/или легитимным участникам, для которых они не предназначены. Легко видеть, что нарушение свойства конфиденциальности ключевой информации (С 8) приводит к нарушению конфиденциальности передаваемых данных.

17) Свойство инвариантности отправителя заключается в том, что на протяжении выполнения всего протокола получатель сообщений сохраняет уверенность в том, что источник сообщения остался тем же, что и источник, с которым было начато взаимодействие (сессия протокола).

18) Свойство анонимности субъекта (участника протокола) состоит в том, что нарушитель, осуществляющий перехват сообщений, не должен иметь возможность связать сообщения одного из участников с самим участником или его идентификатором.

19) Свойство анонимности субъекта для других участников заключается в том, что каждый участник взаимодействия не должен иметь возможность узнать реальную личность других участников, а должен взаимодействовать с их псевдонимом или случайным идентификатором.

20) Свойство защищённости от атак «отказ в обслуживании» подразумевает, что реализующее протокол средство защиты информации обеспечивает алгоритмические, технические и организационно-штатные меры защиты от указанного типа атак. Теоретическое исследование протокола может лишь проверить наличие алгоритмических мер, обеспечивающих защиту от данного класса атак, а также наличие эксплуатационной документации, содержащей описание технических и организационно-штатных мер защиты. В рамках предлагаемой методики представляется возможным получить лишь тривиальное численное значение показателя эффективности для данного свойства.

21) Свойство защищённости от утечек по скрытым (логическим) каналам подразумевает, что протокол содержит реализацию алгоритмических мер защиты от атак, реализуемых нарушителем путём применения непредусмотренных коммуникационных каналов передачи информации. Отметим, что современные транспортные протоколы, такие, как ESP, IPSec или ADTP FIOT, содержат ряд мер, предназначенных для обеспечения данного свойства. Классификация угроз безопасности, реализуемых с использованием скрытых каналов, модель нарушителя и перечень мер защиты информационной системы от атак с использованием скрытых каналов должны разрабатываться на основе стандартов.

22) Свойство защищённости от KCI-атак. Под KCI-атакой (атакой имперсонификации при компрометации долговременного секретного ключа) понимается атака, при выполнении которой нарушитель, получивший доступ к долговременному секретному ключу участника, может выдать себя перед ним за любого другого участника в рамках текущей или будущей сессии выполнения протокола. Свойство считается выполненным, если KCI-атака невыполнима.

23) Свойство защищённости от UKS-атак. Под UKS-атакой понимается последовательность действий нарушителя, в результате которой законные авторизованные участники в процессе информационного взаимодействия вырабатывают общий ключ, но один из участников

считает, что он выработал общий ключ с третьим участником (навязанным нарушителем в ходе выполнения протокола). При этом компрометации общего ключа как таковой не происходит, но нарушается требование аутентификации участников. Свойство считается выполненным, если подобная ситуация невозможна.

24) Свойство невозможности отказа от совершённых действий представляет собой возможность проследить за всеми действиями участника взаимодействия. Согласно Р 1323565.1.012-2017, разд. 6.1.14, данное свойство должно обеспечиваться средством криптографической защиты информации, реализующим криптографический протокол.

25) Свойство доказательства происхождения заключается в неоспоримом доказательстве отправки сообщения.

26) Свойство доказательства доставки заключается в неоспоримом доказательстве получения сообщения.

27) Свойство целостности множества состояний (криптографическое связывание состояний) заключается в том, что все участники информационного взаимодействия после выполнения протокола (или его части) в рамках одного сеанса связи имеют одинаковое представление обо всех участниках этого сеанса и выполняемых ими ролях, а также о состоянии выполнения протокола.

При этом среди них есть базовые свойства (выполнение которых зависит от сложности решения математических задач, используемых в криптографических преобразованиях) и есть те, которые зависят от базовых. Зависимость показана в таблице.

Свойство	Зависимость
С 1 – аутентификации участника протокола другим участником	Базовое
С 2 – аутентификации сообщения	С 1, С 3
С 3 – целостности сообщений	Базовое
С 4 – защиты от повторов	Базовое
С 5 – неявной аутентификации получателя	С 1, С 9
С 6 – групповой аутентификации	С 1, С 9
С 7 – аутентификации субъекта доверенной третьей стороной	С 1
С 8 – конфиденциальности ключа	Базовое
С 9 – аутентификации ключа	С 1, С 2, С 10, С 15
С 10 – подтверждения ключа	Базовое
С 11 – стойкости при компрометации производных ключей	С 13
С 12 – стойкости при компрометации ключа аутентификации	С 13
С 13 – формирования новых ключей	С 15
С 14 – защиты от навязывания ключевых значений	С 1, С 3
С 15 – защиты от навязывания параметров безопасности	С 1, С 2, С 3
С 16 – конфиденциальности	С 3, С 9, С 10
С 17 – инвариантности отправителя	С 1, С 9
С 18 – анонимности субъекта	Базовое
С 19 – анонимности субъекта для других участников	Базовое
С 20 – защищённости от атак «отказ в обслуживании»	Базовое
С 21 – защищённости от утечек по скрытым (логическим) каналам	Базовое
С 22 – защищённости от KCI-атак	С 1, С 9, С 10, С 12, С 13, С 14
С 23 – защищённости от UKS-атак	С 1, С 9, С 10, С 15, С 27
С 24 – невозможности отказа от совершенных действий	С 25, С 26, С 27
С 25 – доказательства происхождения	С 1, С 2, С 9
С 26 – доказательства доставки	С 9, С 10
С 27 – целостности множества состояний	С 1, С 17, С 22, С 23

26. Методы и средства обеспечения заданных свойств безопасности криптографических протоколов.

–

27. Протоколы выработки общего ключа.

Выработка общего ключа – ситуация, когда до конца работы протокола ни у одной из сторон заранее нет ключа, и он появляется только как результат работы протокола. Не путать с распределением ключей – ситуацией, когда ключ генерируется одной из сторон и задачей является «раздать» этот ключ всем остальным.

Классика в выработке общего ключа, которая до сих пор используется с некоторыми изменениями – протокол Диффи-Хеллмана (DH). Протокол предельно прост и включает следующие шаги:

1) A : выбирает числа $g, p,$ а также генерирует случайное число a
2) B : генерирует случайное число b
3) A : вычисляет $X = g^a \ mod \ p$
4) $A \to B : g, p, X$
5) B : вычисляет $Y = g^b \ mod \ p$
6) $B \to A : Y$
7) A : вычисляет $K = Y^a \ mod \ p = g^{ab} \ mod \ p$
8) B : вычисляет $K = X^b \ mod \ p = g^{ab} \ mod \ p$
9) Значение K, выработанное обеими сторонами, является общим секретным ключом

Здесь:

- A и B – Алиса и Боб – стороны информационного обмена.
- p – случайное простое число такое, что $\frac{p-1}{2}$ – тоже простое число.
- g – первообразный корень по модулю p (тоже простое число).

- a и b – большие случайные числа.

В протоколе важно то, что абсолютно все передаваемые значения могут быть доступны злоумышленнику, и это никак не повлияет на безопасность. Однако это работает только в том случае, когда злоумышленник пассивен, т.е. может только прослушивать канал, но не перехватывать и не отправлять сообщения в нём. Т.е. протокол уязвим к атаке «человек посередине».

Чтобы избежать этой уязвимости вводятся модифицированные протоколы. Первый из них – STS – выглядит следующим образом (все предварительные вычисления на сторонах A и B остаются такими же):

1) $A \to B : g^a \ mod \ p$

2) $B \to A : g^b \ mod \ p, E_K(S_B(g^b, g^a))$

3) $A \to B : E_K(S_A(g^a, g^b))$

Здесь:

- S_A и S_B – цифровые подписи сторон A и B.
- $E_K(x)$ – зашифрование сообщения x симметричным алгоритмом с использованием выработанного ключа K.

Также следует отметить, что ключ K становится известен стороне B уже на шаге 2, стороне A – на шаге 3. Следовательно, на шаге 2 Боб уже может зашифровать сообщение на этом ключе, а Алиса, в свою очередь, может расшифровать его на шаге 3.

Цифровые подписи здесь используются для того, чтобы подтвердить, что соответствующие значения пришли именно от той стороны, от которой ожидалось, а не от «человека посередине». Зашифрование используется, чтобы подтвердить, что значение ключа выработано верно.

Также есть модификация под названием MTI. В ней используется вариация криптографии с открытым ключом:

1) A : выбирает секретный ключ $1 \leq a \leq p - 2$

2) A : публикует открытый ключ $Z_A = g^a \ mod \ p$

3) B : выбирает секретный ключ $1 \leq b \leq p - 2$

4) B : публикует открытый ключ $Z_B = g^b \ mod \ p$

5) A : генерирует случайное число $1 \leq x \leq p-2$

6) B : генерирует случайное число $1 \leq y \leq p-2$

7) $A \to B : g^x \ mod \ p$

8) $B \to A : g^y \ mod \ p$

9) Обе стороны вычисляют $K = (g^y)^a Z_B^x \ mod \ p = (g^x)^b Z_A^y \ mod \ p = g^{xb+ya} \ mod \ p$

Публикация открытых ключей приводит к тому, что любая подмена приводит к неверным ключам, и никто ничего не сможет расшифровать.

Также существует модификация DH на эллиптических кривых. В ней всё то же самое за исключением того, что вместо возведения числа в степень по модулю используется умножение точки эллиптической кривой на число.

28. Протоколы распределения ключей.

Распределение ключей применяется в ситуации, когда у одной стороны есть ключ (полученный откуда-то, сгенерированный заранее, либо сгенерированный самостоятельно в начале работы протокола), и этот ключ необходимо «раздать» другим сторонам информационного обмена.

Здесь различают три ситуации:

1) С участием двух сторон при помощи симметричной криптографии

2) С участием трёх сторон (два участника + доверенный центр) при помощи симметричной криптографии

3) При помощи асимметричной криптографии

Здесь и далее будут использоваться следующие обозначения:

- A и B – Алиса и Боб – стороны информационного обмена; таким же образом будут обозначаться уникальные идентификаторы соответствующих сторон
- T – Трент – ещё одна третья (доверенная) сторона информационного обмена, которой доверяют A и B
- K – ключ шифрования

- $E_K(x)$ – зашифрование сообщения x на ключе шифрования K
- $D_K(x)$ – расшифрование шифртекста x на ключе шифрования K
- t – метка времени
- r_X – случайное число, сгенерированное стороной X
- $h_K(x)$ – хеширование с применением ключа шифрования (выработка имитовставки)
- $\oplus$ – побитовый хог
- $S_X(x)$ – цифровая подпись сообщения x закрытым ключом стороны X

28.1. Две стороны и симметричная криптография

В данном случае наиболее простой и наименее реалистичный вариант – когда у двух сторон уже есть общий симметричный ключ, который они каким-то образом получили ранее, и им нужно распределить ещё один.

Тогда можно воспользоваться таким одношаговым протоколом:

$A \to B : E_{K_{AB}}(K, t, B)$

Метка времени здесь необходима, чтобы злоумышленник, перехватив сообщение, не смог подменить этим сообщением аналогичное во время одного из следующих сеансов связи, т.к. в случае компрометации ключа K он мог бы получить доступ к зашифрованным во время этого сеанса данных. Идентификатор принимающей стороны нужен, чтобы злоумышленник не смог использовать это сообщение для отправки обратно передающей стороне во время одного из следующих сеансов связи, когда эта сторона, возможно, уже будет принимающей.

Также есть вариация с использованием имитовставки вместо шифрования:

$A \to B : K \oplus h_{K_{AB}}(t, B)$

Вместо метки времени может использоваться случайное число, сгенерированное принимающей стороной.

А. Шамир (S из RSA) также предложил «бесключевой» протокол, позволяющий двум сторонам передать ключ, не имея какой либо общей секретной информации заранее. Для этого необходимо коммутирующее шифрующее преобразование $E : E_{K_1}(E_{K_2}(x)) = E_{K_2}(E_{K_1}(x))$, где K_1 и K_2 – два разных ключа шифрования.

Тогда Алиса может передать Бобу секретный ключ K следующим образом:

1) $A \to B : E_{K_A}(K)$

2) $B \to A : E_{K_B}(E_{K_A}(K))$

3) $A \to B : D_{K_A}(E_{K_B}(E_{K_A}(K))) = E_{K_B}(K)$

28.2. Три стороны стороны и симметричная криптография

В трёхсторонних протоколах предполагается наличие некоторой третьей доверенной стороны T, у которой уже есть согласованные ключи с каждой из других сторон.

Один из первых протоколов состоит в следующем:

1) $A \to T : A, B, r_A$

2) $T \to A : E_{K_{AT}}(r_A, B, K, E_{K_{BT}}(K, A))$

3) $A \to B : E_{K_{BT}}(K, A)$

4) $B \to A : E_K(r_B)$

5) $A \to B : E_K(r_B - 1)$

Где K_{XY} – симметричный ключ сторон X и Y.

Проблема этого протокола состоит в том, что сообщение, переданное на шаге 3, злоумышленник может передать снова в одном из следующих сеансов связи, что станет проблемой, если переданный в этом сообщении ключ уже был скомпрометирован.

Эта проблем решается в довольно известном протоколе Kerberos. Этот протокол использует две специальные криптографические сущности: «билет» и аутентификатор.

Чтобы пообщаться с Бобом, Алиса должна получить у Трента «билет» на такое общение. Далее, чтобы начать общаться с Бобом, Алиса должна предъявить ему свой «билет» и аутентификатор. Аутентификатор Алиса может генерировать сама, «билет» может сгенерировать только Трент.

Билет выглядит следующим образом: $E_{K_{BT}}(K, A, L)$. Здесь: K_{BT} – ключ Боба и Трента, который они знают заранее, L – время действия билета, K – сеансовый ключ для общения Алисы с Бобом. Важно, что, несмотря на то, что билет выдаётся Алисе, расшифровать она его не может, т.к. он зашифрован ключом, который знают только Трент и Боб.

Таким образом, она может только предъявить билет Бобу как есть, т.е. в зашифрованном виде.

Аутентификатор выглядит так: $E_K(A, t, K_A)$, где K_A – секрет Алисы, который потом может быть использован для создания общего ключа. K_A – необязательный параметр, т.к. у Алисы и Боба к тому моменту уже есть сеансовый ключ, сгенерированный для них Трентом.

Базовый протокол Kerberos содержит следующие шаги:

1) $A \to T : A, B, r_A$

2) $T \to A : E_{K_{AT}}(K, r_A, L, B), \underbrace{E_{K_{BT}}(K, A, L)}_{\text{билет}}$

3) $A \to B : \underbrace{E_{K_{BT}}(K, A, L)}_{\text{билет}}, \underbrace{E_K(A, t, K_A)}_{\text{аутентификатор}}$

4) $B \to A : E_K(t, K_B)$, где K_B – необязательный секрет Боба

По окончании работы протокола Алиса и Боб могут сгенерировать новый общий секретный ключ из секретов K_A и K_B.

Это был только базовый протокол. На практике используется два доверенных сервера. Один выдаёт билеты на получение билетов от второго, а второй уже выдаёт билеты на общение с нужным участником.

Есть ещё много протоколов, использующих симметричную криптографию и доверенную сторону: «Лягушка с широким ртом», Yahalom, Needham-Schroeder, Otway-Rees, Neuman-Stubblebine.

28.3. Две стороны стороны и асимметричная криптография

Здесь в записях вида E_X под X всегда подразумевается открытый ключ, т.к. зашифрование возможно только с его помощью.

Наиболее простая вариация содержит всего один шаг:

$A \to B : E_{K_B}(K, t, A)$

Если нужна взаимная аутентификация, можно воспользоваться следующим протоколом:

1) $A \to B : E_{K_B}(K_1, A)$

2) $B \to A : E_{K_A}(K_1, K_2)$

3) $A \to B : E_{K_B}(K_2)$

Далее из секретов K_1 и K_2 может быть выработан общий секретный ключ. Пересылая эти секреты несколько раз, стороны убеждаются, что они верно расшифровали значение секрета, и что общаются они с нужным участником.

Также для обмена ключами может использоваться цифровая подпись (что обычно и делается в современных протоколах). Например подпись может быть использована одним из следующих способов:

- $A \to B : E_{K_B}(K, t, S_A(B, K, t))$
- $A \to B : E_{K_B}(K, t), S_A(B, K, t)$
- $A \to B : t, E_{K_B}(A, K), S_A(B, t, E_{K_B}(A, K)))$

Также, например, Шнайер приводит много протоколов, использующих подписи и доверенную сторону: DASS, Denning-Sacco, Woo-Lam.

29. Протоколы с разделением секрета.

Разделение секрета – ситуация, когда секрет (как правило, ключ) хранится по частям у нескольких сторон и только некоторое подмножество этих сторон может восстановить его.

Это применяется для достижения двух целей: защиты от утери секрета и разделения ответственности за какие-либо важные действия с секретом (например, если ключ нужен для запуска какого-то оружия и т.п.).

Простейший вариант такой схемы – когда секрет необходимо разделить между n сторонами, и, только собравшись вместе, они должны иметь возможность его восстановить. Тогда можно применить следующий подход (предполагается, что секрет – некоторый двоичный вектор или представим в таком виде): генерируется n двоичных векторов так, чтобы они при сложении (побитовым xor) давали исходный секрет. Каждый из векторов раздаётся сторонам обмена. Таким образом, получается, что только сума всех векторов может дать результирующий секрет, тогда как любое их подмножество само по себе бесполезно.

Важно отметить, что в таком простом случае не подойдёт вариант просто разбить вектор на n частей и отдать участникам, т.к. каждая из этих частей будет нести некоторую информацию об исходном секрете (тогда, например, собрав $n - 1$ частей, оставшуюся часть можно будет подобрать методом грубой силы), тогда как в предыдущем варианте каждый из векторов не раскрывает о секрете абсолютно ничего.

Такая схема достаточно безопасна, но неудобна. На практике чаще возникают ситуации, когда нужно разделить секрет между n участниками так, чтобы любые $m < n$ из них могли его восстановить. Алгоритм разделения ключа, удовлетворяющий описанному условию, называется (m, n)-пороговой схемой.

Существует несколько реализаций таких схем, основанных на свойствах различных математических объектов. Здесь будет приведён один из наиболее простых подходов, предложенный Шамиром.

Чтобы реализовать (m, n)-пороговую схему при таком подходе:

1) Генерируется простое число p, большее n и большее значения самого большого возможного секрета.

2) Генерируется произвольный многочлен степени $m - 1$ вида

$$F(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + M$$

где M – разделяемый секрет. Коэффициенты a_i хранятся в секрете и отбрасываются после распределения частей секрета.

3) В качестве частей секрета вычисляются значения многочлена в различных точках, например, $k_1 = F(1)$, $k_2 = F(2)$ и т.д.

4) Далее, чтобы восстановить M, необходимо решить систему из m уравнений вида

$$\begin{cases} \underbrace{a_{m-1} \cdot 1^{m-1} + a_{m-2} \cdot 1^{m-2} + ... + a_1 \cdot 1 + M}_{F(1)} = k_1, \\ F(2) = k_2, \\ ..., \\ F(i) = k_i, \\ ... \end{cases}$$

Очевидно, что, если собрать меньше, чем m частей, уравнение не решится.

В этой схеме можно легко увеличить n, просто сгенерировав значения многочлена в новых точках.

Более того, как утверждает Шнайер, если a_i выбраны случайно, то даже с бесконечными вычислительными мощностями злоумышленник никак не сможет получить секрет M, не собрав хотя бы m его частей.

30. Протоколы с подписью вслепую и протоколы электронного голосования.

–

31. Протоколы семейства TLS, область их применения, методы оценки безопасности.

TLS (англ. transport layer security — Протокол защиты транспортного уровня) – криптографический протокол, обеспечивающий защищённую передачу данных между узлами в сети Интернет.

Использует асимметричное шифрование для аутентификации, симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений.

Область применения: прежде всего, HTTPS, т.е. практически все веб-браузеры и сайты соответственно. Однако также используется в электронной почте, чатах и IP-телефонии.

В общем все версии протокола действуют по следующему алгоритму:

1) Клиент подключается к серверу и отправляет ему поддерживаемую версию протокола TLS и поддерживаемые алгоритмы шифрования.

2) Сервер отвечает, какую версию и какое шифрование он готов использовать.

3) Сервер отправляет сертификат своего открытого ключа, если разделение общего ключа будет происходить через асимметричную криптографию.

4) Сервер отправляет свою часть Диффи-Хеллмана, если разделение общего ключа будет происходить через DH.

5) Клиент, в зависимости от способа разделения ключа, отправляет либо свою часть DH, либо зашифрованный на открытом ключе сервера секрет.

6) При необходимости клиент и сервер завершают выработку общего секрета, на основе которого будет осуществляться дальнейшее шифрование всего.

7) Клиент сообщает, что дальнейшая связь будет зашифрована и в зашифрованном виде отправляет хеш и имитовставку всех предыдущих сообщений.

8) Сервер всё проверяет и отправляет то же самое.

9) Клиент всё проверяет.

10) Если все проверки успешны, соединение считается установленным и дальше вся передача данных зашифровывается на основе разделённого секрета.

От версии к версии в протоколе меняются алгоритмы шифрования, цифровой подписи и выработки имитовставки, рекомендованные к использованию.

В частности, TLS 1.3 (последняя на данный момент версия) содержит алгоритмы из ГОСТ.

32. Протокол SSH, область его применения, реализуемые методы аутентифицированного удаленного доступа.

–

33. Протокол SESPAKE выработки общего ключа на основе пароля, область его применения, принципы обоснования сложности перебора паролей.

SESPAKE по сути является большим расширением над протоколом Диффи-Хеллмана, позволяющим защитить выработку общего ключа с помощью слабого секрета – пароля.

Сам протокол описан в Рекомендациях по стандартизации Р 50.1.115-2016 «Протокол выработки общего ключа с аутентификацией на основе пароля», включает 27 шагов и большое количество различных вспомогательных значений, так что здесь будут описаны только основные принципы, лежащие в его основе.

В ходе протокола Диффи-Хеллмана две стороны открыто согласуют два числа p и g, после чего отправляют друг другу значения $g^a \ mod \ p$ и $g^b \ mod \ p$, где a – сгенерированное одной стороной секретное случайное число, b – аналогичное число другой стороны. Потом каждая сторона вычисляет $(g^a)^b \ mod \ p = (g^b)^a \ mod \ p$ и принимает полученное значение за секретный ключ. Известная проблема данного протокола состоит в том, что он уязвим к атаке «человек посередине», когда активный злоумышленник вторгается в линию связи и может отменять пересылку некоторых сообщений и заменять их на другие.

Чтобы избежать описанной проблемы, можно применить аутентификацию на основе пароля. Для этого необходимо, чтобы каждый из участников заранее знал некоторый пароль S, одинаковый для обоих участников. Тогда протокол Диффи-Хеллмана можно дополнить пересылкой сообщений вида:

$$A \to B : H(S||K_A||1)$$
$$B \to A : H(S||K_B||2)$$

Где K_A и K_B – ключи, полученные сторонами в результате выполнения протокола Диффи-Хеллмана, и, если злоумышленник не вторгся в процесс работы, $K_A = K_B$.

Такой подход позволяет сторонам убедиться, что они получили один и тот же ключ, причём получили его именно они и никто больше.

Однако, если учесть, что S – пароль, то есть некоторая легко перебираемая машинно величина, можно увидеть, что становится возможной следующая атака. Злоумышленник может подменить собой сторону B и участвовать в протоколе до тех пор, пока не получит $H(S||K_A||1)$ от A, после чего прервать связь. Т.к. злоумышленник сам участвовал в выработке ключа, он знает K_A, а значит, $H(S||K_A||1)$ даёт ему критерий подбора пароля. Т.е. теперь злоумышленник может без какой-либо связи с легитимными сторонами взаимодействия подставлять в хеш-функцию все подбираемые значения паролей, дополнять их ключом K_A и значением 1 и проверять, верный ли получился хеш. Когда станет верный, злоумышленник подобрал пароль. Такая атака называется offline dictionary attack.

Чтобы такого не происходило, нужно включить пароль в сам процесс генерации ключа. Именно на этой идее основаны PAKE-протоколы (PAKE – Password Authenticated Key Exchange).

Для реализации простейшего из таких протоколов стороны должны согласовать и открыто опубликовать ещё одно число – h. При этом важно, чтобы дискретные логарифмы h по основанию g и наоборот были

неизвестны, т.е. h и g должны быть сгенерированы случайно. Протокол выглядит следующим образом:

1) $A \to B : M = g^a h^S$

2) $B \to A : N = g^b h^S$

3) A : вычисляет $K_A = (N/h^S)^a$

4) B : вычисляет $K_B = (N/h^S)^b$

5) $A \to B : H(K_A||1)$

6) $B \to A : H(K_B||2)$

При таком подходе злоумышленник уже не может подменить собой Боба, т.к. не знает пароль, а чтобы получить критерий для перебора пароля, ему нужно будет как-то вычислить K_A, что является вычислительно сложной задачей, лежащей в основе протокола Диффи-Хеллмана.

Однако протокол можно сделать ещё безопаснее. Можно сделать так, чтобы, даже зная K_A и K_B злоумышленник не смог получить доступ к паролю. Для этого достаточно ввести хеширование на шагах 3 и 4, т.е., например, шаг 3 станет выглядеть следующим образом:

$$A \text{ : вычисляет } K_A = H((N/h^S)^a)$$

На таких принципах построен протокол SESPAKE. Однако в нём используется версия Диффи-Хеллмана на эллиптических кривых вместо классической, а также много дополнительных значений и шагов для ещё большего повышения безопасности.

Применяется этот протокол, в основном, при взаимодействии с функциональными ключевыми носителями (ФКН). ФКН – специальные активные устройства для хранения ключа, которые никогда не передают сам ключ по каналу связи, но при этом могут сами выполнить необходимые действия с использованием ключа, например, подписать документ. Выглядеть это может, например, как флешка или смарт-карта, в которую встроены собственные процессор, память и т.д.

Без протокола SESPAKE пользователю приходилось передавать пароль на это устройство, после чего оно могло выполнить необходимую операцию, но злоумышленник в канале передачи мог получить доступ к паролю. С протоколом SESPAKE устройство соединяется со специальным ПО для ввода пароля, которое отрабатывает протокол и, соответственно, препятствует передаче пароля или какой-либо информации о нём по каналу связи.

34. Протокол защищенного взаимодействия SP-FIOT. Обоснование свойств безопасности, отличия от других протоколов.

–

35. Криптографические механизмы протокола IPSec. Обеспечиваемые им свойства безопасности.

–