11. Понятие надежности. Методология обоснования надежности криптографической защиты.

Надёжность = стойкость.

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 зашифровыва-

ется 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.

–

19. Ключевые системы, методы распределения ключей.

–

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

–

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 сер-

тификаты – 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.

–

25. Свойства безопасности криптографических протоколов.

–

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

–

27. Протоколы выработки общего ключа.

–

28. Протоколы распределения ключей.

–

29. Протоколы с разделением секрета.

–

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.

–

34. Протокол защищенного взаимодействия SPFIOT. Обоснование свойств безопасности, отличия от других протоколов.

–

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

–