ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

РЕКОМЕНДАЦИИ ПО СТАНДАРТИЗАЦИИ

Р 1323565.1.028— 2019

Информационная технология

КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ

Криптографические механизмы защищенного взаимодействия контрольных и измерительных устройств

Издание официальное

Москва
Стандартинформ
2020

Предисловие

1 РАЗРАБОТАНЫ Обществом с ограниченной ответственностью Фактор-ТС (ООО Фактор-ТС)

2 ВНЕСЕНЫ Техническим комитетом по стандартизации ТК 26 «Криптографическая защита информации»

3 УТВЕРЖДЕНЫ И ВВЕДЕНЫ В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 30 декабря 2019 г. № 1503-ст

4 ВВЕДЕНЫ ВПЕРВЫЕ

Правила применения настоящих рекомендаций установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящим рекомендациям публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящих рекомендаций соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.gost.ru)

Содержание

Введение

Настоящие рекомендации определяют криптографические механизмы защищенного взаимодействия между двумя абонентами по незащищенному каналу связи.

Определяемые механизмы могут быть отнесены к сеансовому и транспортному уровням модели взаимосвязи открытых систем (далее — ВОС) согласно ГОСТ Р ИСО/МЭК 7498-1 и предназначены для аутентификации взаимодействующих абонентов, а также обеспечения целостности и, при необходимости, конфиденциальности передаваемой информации.

Механизмы формирования и обработки информации на прикладном уровне модели ВОС в настоящем документе не рассматриваются. Защищенное взаимодействие на сеансовом уровне осуществляется путем выполнения сеансов связи. В ходе каждого сеанса связи происходит последовательное выполнение двух криптографических протоколов:

протокола выработки ключей, предназначенного для аутентификации взаимодействующих абонентов и выработки общей ключевой информации, используемой для обеспечения целостности и конфиденциальности передаваемой информации;

протокола передачи прикладных данных, в рамках которого происходит взаимодействие между абонентами на прикладном уровне.

Каждый из указанных протоколов использует единый транспортный протокол для отправки и получения информации из канала связи. Схема информационного обмена в ходе организации защищенного взаимодействия приведена на рисунке 1.

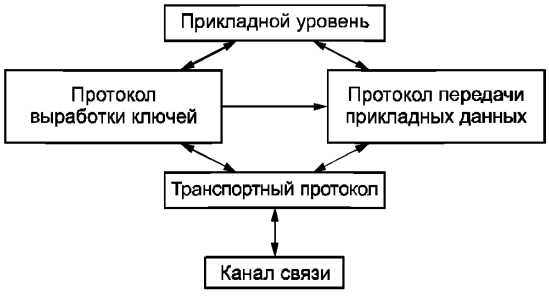

Рисунок 1 — Схема информационного обмена

Допускается использование каналов связи, для которых свойство гарантированной доставки пакетов не обеспечивается.

РЕКОМЕНДАЦИИ ПО СТАНДАРТИЗАЦИИ

Информационная технология

КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ

Криптографические механизмы защищенного взаимодействия контрольных и измерительных устройств

Information technology. Cryptographic data security. Cryptographic mechanisms of secure interactions of control and measuring devices

Дата введения — 2020—09—01

1 Область применения

Настоящие рекомендации предназначены для обеспечения защищенного взаимодействия между двумя абонентами по незащищенному каналу связи, а также для реализации каналов удаленного управления.

В качестве абонентов могут выступать контрольные и измерительные устройства, объекты «Интернета вещей», миниатюрные технические составляющие различных технологических процессов, обменивающиеся служебной информацией, а также произвольные субъекты автоматизированных систем, для которых необходим защищенный обмен информацией, не содержащей сведений, составляющих государственную тайну.

Описываемые в настоящем документе механизмы могут применяться в средствах криптографической защиты информации (далее — СКЗИ) всех классов, определяемых Р 1323565.1.012—2017.

Соответствие изложенным в Р 1323565.1.012—2017 принципам позволяет реализовать механизм регулярного изменения ключевой информации, используемой для обеспечения целостности и, при необходимости, конфиденциальности передаваемой информации.

Изменение ключевой информации в ходе одного сеанса связи позволяет передавать большие объемы информации без установления нового сеанса связи. Это может быть востребовано устройствами, длительное время функционирующими без взаимодействия с человеком. При этом процедура установления сеанса связи может выполняться при начальной инициализации таких устройств.

Значения параметров, определяющих максимальный объем информации, которая может быть передана в ходе одного сеанса связи, приводятся в приложении Г.

2 Нормативные ссылки

В настоящих рекомендациях использованы нормативные ссылки на следующие стандарты:

ГОСТ Р 34.10—2012 Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи

ГОСТ Р 34.11—2012 Информационная технология. Криптографическая защита информации. Функция хэширования

ГОСТ Р 34.12—2015 Информационная технология. Криптографическая защита информации. Блочные шифры

ГОСТ Р 34.13—2015 Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров

ГОСТ Р ИСО/МЭК 7498-1 Информационная технология. Взаимосвязь открытых систем. Базовая эталонная модель. Часть 1. Базовая модель

Издание официальное

Р 50.1.113—2016 Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов электронной цифровой подписи и функции хэширования

Р 1323565.1.004—2017 Информационная технология. Криптографическая защита информации. Схемы выработки общего ключа с аутентификацией на основе открытого ключа

Р 1323565.1.005—2017 Информационная технология. Криптографическая защита информации. Допустимые объемы материала для обработки на одном ключе при использовании некоторых вариантов режимов работы блочных шифров в соответствии с ГОСТ Р 34.13—2015

Р 1323565.1.012—2017 Информационная технология. Криптографическая защита информации. Принципы разработки и модернизации шифровальных (криптографических) средств защиты информации

Р 1323565.1.017—2018 Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов блочного шифрования

Р 1323565.1.019—2018 Информационная технология. Криптографическая защита информации. Криптографические механизмы аутентификации и выработки ключа фискального признака для применения в средствах формирования и проверки фискальных признаков, обеспечивающих работу контрольно-кассовой техники, операторов и уполномоченных органов обработки фискальных данных

Р 1323565.1.023—2018 Информационная технология. Криптографическая защита информации. Использование алгоритмов ГОСТ Р 34.10—2012, ГОСТ Р 34.11—2012 в сертификате, списке аннулированных сертификатов (CRL) и запросе на сертификат PKCS # 10 инфраструктуры открытых ключей X.509

Р 1323565.1.024—2019 Информационная технология. Криптографическая защита информации. Параметры эллиптических кривых для криптографических алгоритмов и протоколов

Р 1323565.1.026—2019 Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров, реализующие аутентифицированное шифрование

Примечание — При пользовании настоящими рекомендациями целесообразно проверить действие ссылочных стандартов в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если заменен ссылочный стандарт, на который дана недатированная ссылка, то рекомендуется использовать действующую версию этого стандарта с учетом всех внесенных в данную версию изменений. Если заменен ссылочный стандарт, на который дана датированная ссылка, то рекомендуется использовать версию этого стандарта с указанным выше годом утверждения (принятия). Если после утверждения настоящих рекомендаций в ссылочный стандарт, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение рекомендуется применять без учета данного изменения. Если ссылочный стандарт отменен без замены, то положение, в котором дана ссылка на него, применяется в части, не затрагивающей эту ссылку.

3 Термины и определения

В настоящих рекомендациях применены следующие термины с соответствующими определениями.

3.1 **абонент:** Участник информационного, в том числе защищенного криптографическими методами, сетевого взаимодействия.

3.2 **действительный сертификат открытого ключа; СОК:** Сертификат ключа проверки электронной подписи, срок действия которого не истек, электронная подпись которого проверяется корректно, а сфера применения сертификата допускает его использование для аутентификации и выработки общего ключа.

3.3 **клиент:** Абонент, являющийся обслуживаемой стороной информационного взаимодействия, инициирующий выполнение защищенного взаимодействия.

3.4 **ключ проверки электронной подписи:** Уникальная последовательность символов, однозначно связанная с ключом электронной подписи и предназначенная для проверки подлинности электронной подписи.

3.5 **ключ электронной подписи:** Уникальная последовательность символов, предназначенная для создания электронной подписи.

3.6 **октет:** Символ, который может быть представлен в виде двоичной последовательности длины восемь.

3.7 **опциональная последовательность:** Последовательность октетов, значение которой может быть определено и присутствовать в передаваемых по каналам связи данных, либо не определено и отсутствовать в передаваемых по каналам связи данных.

3.8 **сеанс связи:** Полный цикл информационного обмена между абонентами, подразумевающий установление соединения, согласование параметров связи, в том числе параметров защиты информации, передачу и/или получение данных или команд, а также закрытие соединения.

3.9 **сервер:** Абонент, являющийся источником установления соединения в информационном взаимодействии, отвечающий на запрос клиента.

3.10 **сериализация:** Процесс перевода структуры данных в последовательность октетов конечной длины.

3.11 **сертификат ключа проверки электронной подписи:** Электронный документ или документ на бумажном носителе, выданный удостоверяющим центром либо доверенным лицом удостоверяющего центра и подтверждающий принадлежность ключа проверки электронной подписи владельцу сертификата ключа проверки электронной подписи.

3.12 **удостоверяющий центр:** Юридическое лицо или индивидуальный предприниматель, осуществляющий функции по созданию и выдаче сертификатов ключей проверки электронной подписи, а также иные функции, предусмотренные [1].

3.13 **электронная подпись:** Информация в электронной форме, которая присоединена к другой информации в электронной форме (подписываемой информации) или иным образом связана с такой информацией и которая используется для определения лица, подписывающего информацию.

3.14 **эллиптическая кривая:** Множество решений уравнения специального вида, образующее конечную группу, в которой реализуется протокол выработки общих ключей.

3.15 **фрейм:** Последовательность октетов, имеющая внутреннюю логическую структуру и являющаяся единицей обмена информацией между абонентами.

4 Обозначения

В настоящих рекомендациях использованы следующие обозначения:

V^* — множество всех двоичных последовательностей конечной длины, включая последовательность нулевой длины;

V^s — множество всех двоичных последовательностей, состоящих в точности из s элементов;

B^* — множество всех последовательностей октетов конечной длины, включая последовательность нулевой длины;

B^s — множество всех последовательностей октетов, состоящих в точности из s октетов; последовательность октетов $o \in B^s$ записывается в виде $o = (o_0, \dots, o_{s-1})$, где каждая из координат $o_0, \dots, o_{s-1}$ принадлежит множеству V^8;

Len(s) — функция, возвращающая в качестве значения длину последовательности октетов $o \in B^s$, т. е. для $o = (o_0, \dots, o_{s-1})$ выполнено равенство $Len(o) = s$;

o[n] — операция выбора из последовательности октетов $o \in B^s$ заданного октета с индексом n, т. е. для $o = (o_0, \dots, o_{s-1})$, где $0 \le n < s$, выполнено равенство $o[n] = o_n$;

o[n, .., m] — для целых чисел $0 \le n \le m$ операция выбора из последовательности октетов $o \in B^s$ подпоследовательности, начинающейся с индекса n и заканчивающейся индексом m, т. е. для $o = (o_0, \dots, o_n, \dots, o_m, \dots, o_{s-1})$ выполнено равенство $o[n, \dots, m] = (o_n, \dots, o_m)$;

$Msb_n(o)$ — функция, возвращающая подпоследовательность октетов, состоящую из n октетов со старшими номерами последовательности октетов $o \in B^s$, т. е. для $o = (o_0, \dots, o_{s-1})$ выполнено равенство $Msb_n(o) = o[s-n, \dots, s-1]$;

Ser(t) — последовательность октетов конечной длины, являющаяся результатом сериализации одной из структур данных, определяемых в приложении В;

p, q — простые числа;

Fp^n — для натурального числа n и простого числа p конечное поле из p^n элементов;

$F_{2^n}[x]$ — кольцо многочленов от одной переменной с коэффициентами из конечного поля F_{2^n};

$F_{2^n}[x, y]$ — кольцо многочленов от двух переменных x и y с коэффициентами из конечного поля F_{2^n};

E — эллиптическая кривая, определенная над полем F_p в канонической форме Вейерштрасса, либо в форме скрученной кривой Эдвардса, см. Р 1323565.1.024—2019;

a, b — элементы поля F_p, являющиеся коэффициентами эллиптической кривой, заданной в канонической форме Вейерштрасса сравнением $y^2 \equiv x^3 + ax + b \pmod p$;

e, d — элементы поля F_p, являющиеся коэффициентами эллиптической кривой, заданной в форме скрученной кривой Эдвардса сравнением $eu^2 + v^2 \equiv 1 + du^2v^2 \pmod p$;

O	— нейтральный элемент группы точек эллиптической кривой E (бесконечно удаленная точка);
P,Q	— точки эллиптической кривой E, определяемые парой координат (элементов поля F_p), либо парой (x,y) в случае кривой, заданной в канонической форме Вейерштрасса, либо парой (u,v) в случае кривой, заданной в форме скрученной кривой Эдвардса;
x(Q)	— функция, возвращающая в качестве значения первый элемент пары координат, определяющей точку Q эллиптической кривой, либо x, для точки принадлежащей кривой, заданной в канонической форме Вейерштрасса, либо u, для точки принадлежащей кривой, заданной в форме скрученной кривой Эдвардса;
y(Q)	— функция, возвращающая в качестве значения второй элемент пары координат, определяющей точку Q эллиптической кривой, либо y, для точки принадлежащей кривой, заданной в канонической форме Вейерштрасса, либо v, для точки принадлежащей кривой, заданной в форме скрученной кривой Эдвардса;
ID_E	— идентификатор эллиптической кривой, используемый в ходе выполнения протокола выработки ключей;
ID_C, ID_S	— последовательности октетов, определяющие идентификаторы, соответственно, клиента и сервера;
ID_{iPSK}, ID_{ePSK}	— идентификаторы предварительно распределенных ключей аутентификации, см. 5.2;
E(K,M)	— алгоритм зашифрования одного блока информации M на ключе K с помощью алгоритма блочного шифрования; перечень допустимых алгоритмов блочного шифрования определяется в В.3.8;
Enc(K,M,I)	— алгоритм зашифрования (режим работы блочного шифра) сообщения M произвольной длины на ключе K с использованием синхропосылки I; перечень допустимых алгоритмов зашифрования определяется в В.3.8;
Dec(K,M)	— алгоритм расшифрования (режим работы блочного шифра) сообщения M произвольной длины с помощью ключа K; перечень допустимых алгоритмов расшифрования определяется в В.3.8;
$\{M\}_K$	— обозначение алгоритма зашифрования сообщения M произвольной длины на ключе K, используемое на содержащихся в настоящих рекомендациях рисунках; в случае, если значение ключа является не существенным, обозначение K может быть на рисунке не указано;
$Streebog_n$	— регламентируемая ГОСТ Р 34.11—2012 бесключевая функции хэширования «Стрибог» с длиной блока n бит, где величина n принимает значение 256 либо 512;
[M]	— обозначение результата применения произвольной бесключевой функции хэширования к сообщению M, используемое на содержащихся в настоящих рекомендациях рисунках;
$HMAC_n$	— алгоритм выработки имитовставки HMAC с длиной кода n бит, регламентируемый рекомендациями Р 50.1.113—2016;
Mac(K,M,I)	— алгоритм выработки имитовставки под сообщением M на ключе K с использованием синхропосылки I (использование синхропосылки является опциональным); перечень допустимых алгоритмов выработки имитовставки определяется в В.3.8;
$[M]_K$	— обозначение алгоритма выработки имитовставки под сообщением M на ключе K, используемое на содержащихся в настоящих рекомендациях рисунках;
$Cert_c$, $Cert_s$	— сертификат ключа проверки электронной подписи клиента и, соответственно, сервера;
Sign(d,M)	— алгоритм выработки электронной подписи под сообщением M с помощью ключа электронной подписи d; перечень допустимых алгоритмов выработки электронной подписи определяется в В.3.8;
Veri(Q,M,s)	— алгоритм проверки электронной подписи s под сообщением M с помощью ключа проверки электронной подписи Q; перечень допустимых алгоритмов проверки электронной подписи определяется в В.3.8;
True, False	— булевы переменные, принимающее значения, соответственно, «истина» и «ложь», и являющиеся результатом проверки электронной подписи и/или проверки совпадения имитовставки;
Validate(x)	— функция, проверяющая действительность сертификата x ключа проверки электронной подписи и возвращающая «истину» в случае действительности сертификата и «ложь» в противном случае.

5 Протокол выработки ключей

5.1 Общие сведения

Цель протокола выработки ключей заключается в установлении соединения и выработке клиентом и сервером общей ключевой информации, предназначенной для зашифрования, расшифрования и контроля целостности передаваемой на прикладном уровне информации. В основу протокола положена схема выработки общего ключа с аутентификацией на основе открытого ключа «Эхинацея-3», описываемая Р 1323565.1.004—2017.

Выполнение протокола выработки ключей инициируется клиентом и состоит из четырех этапов, в ходе которых клиент и сервер обмениваются сообщениями. На каждом этапе один из участников протокола формирует, отправляет и/или получает одно или несколько сообщений по незащищенному каналу связи. При этом отправляемые абонентами сообщения могут передаваться как в незашифрованном, так и зашифрованном виде.

Схема формирования, проверки и обмена сообщениями в ходе выполнения протокола выработки ключей представлена на рисунке 2.

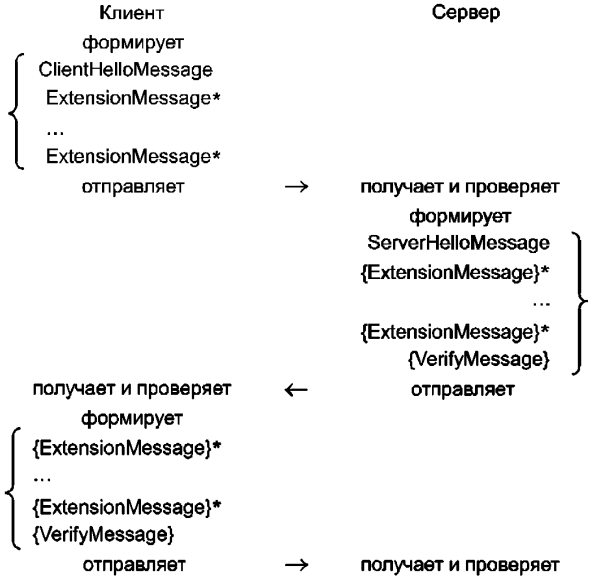

Рисунок 2 — Обмен сообщениями в ходе выполнения протокола выработки ключей

Примечание — На рисунке 2 сообщения, которые являются опциональными, отмечены знаком «*»; сообщения, которые передаются в зашифрованном виде, заключены в фигурные скобки; ExtensionMessage обозначает одно из расширений, определяемых в В.6.

В случае возникновения ошибки абонент (клиент или сервер), обнаруживший ошибку, отправляет сообщение AlertMessage, после чего следует реакция, определяемая порядком поведения абонента при получении сообщения об ошибке, см. В.5.5.

Протокол выработки ключей реализуется с использованием арифметических операций в группе точек эллиптической кривой E, удовлетворяющей требованиям, предъявляемым к эллиптическим кривым в ГОСТ Р 34.10—2012, раздел 5, и являющейся параметром протокола. Для реализации протокола выработки ключей рекомендуется использование эллиптических кривых, определяемых как в канонической форме, так и в форме скрученных кривых Эдвардса, см. В.3.9.

В ходе выполнения протокола в обязательном порядке происходит аутентификация сервера, а также по желанию сервера аутентификация клиента. Аутентификация абонентов может реализовываться как с использованием механизма электронной подписи, так и с использованием механизма предварительно распределенных симметричных ключей аутентификации.

Результатом выполнения протокола является набор ключей шифрования и выработки имитовставки, используемых абонентами в ходе выполнения протокола передачи прикладных данных для обеспечения конфиденциальности и целостности передаваемой информации.

Описываемый протокол выработки ключей представляет собой объединение нескольких подходов к выработке общей ключевой информации, основанных на различных принципах аутентификации абонентов. Различные типы контрольных и измерительных устройств, в силу своих технических и эксплуатационных особенностей, могут следовать рекомендуемой последовательности действий, но реализовывать при этом только один из возможных принципов аутентификации. В приложении А приводятся типовые схемы реализации протокола выработки ключей.

5.2 Ключевая система

Ключевая система протокола выработки ключей состоит из следующего множества:

ключей аутентификации, предназначенных для аутентификации абонентов;

общей ключевой информации;

ключей шифрования и выработки имитовставки, предназначенных для обеспечения конфиденциальности и целостности информации, передаваемой в ходе выполнения протокола выработки ключей.

В результате выполнения протокола вырабатывается ключевая информация, используемая в ходе выполнения протокола передачи прикладных данных для выработки производных ключей шифрования и выработки имитовставки.

5.2.1 Ключи аутентификации

В протоколе выработки ключей могут быть использованы следующие ключи аутентификации:

$d_c \in F_q$ — ключ электронной подписи клиента, удовлетворяющий ГОСТ Р 34.10—2012, раздел 5;

$Q_c \in E$ — ключ проверки электронной подписи клиента, математически связанный с ключом электронной подписи d_c и удовлетворяющий ГОСТ Р 34.10—2012, раздел 5; корректность ключа проверки электронной подписи клиента и его однозначное соответствие идентификатору клиента ID_c должны гарантироваться сертификатом $Cert_c$ ключа проверки электронной подписи;

$d_s \in F_q$ — ключ электронной подписи сервера, удовлетворяющий ГОСТ Р 34.10—2012, раздел 5;

$Q_s \in E$ — ключ проверки электронной подписи сервера, математически связанный с ключом электронной подписи сервера d_s и удовлетворяющий ГОСТ Р 34.10—2012, раздел 5; корректность ключа проверки электронной подписи сервера и его однозначное соответствие идентификатору сервера ID_s должны гарантироваться сертификатом $Cert_s$ ключа проверки электронной подписи;

$ePSK \in B^s$ — предварительно распределенный, общий для клиента и сервера ключ аутентификации длиной s октетов, где $s \in \{32, \ldots, 64\}$; ключ ePSK является опциональным;

$iPSK \in B^{32}$ — общий для клиента и сервера ключ аутентификации; вырабатывается в ходе выполнения протокола передачи прикладных данных и может использоваться в ходе выполнения последующего сеанса протокола выработки ключей.

Алгоритм выработки ключа аутентификации iPSK и связывания данного ключа с идентификатором ID_{iPSK} описывается в 6.6. Рекомендации по выработке предварительно распределенных ключей приводятся в приложении Б.

Способы выработки и доведения до абонентов ключей электронной подписи, ключей проверки электронной подписи и сертификатов ключей проверки электронной подписи в настоящих рекомендациях не рассматриваются.

5.2.2 Общая ключевая информация

В ходе выполнения протокола клиентом выбирается эллиптическая кривая E из множества эллиптических кривых, идентификаторы которых определены в В.3.9. Данная кривая характеризуется следующими параметрами:

выделенной точкой эллиптической кривой P;

простым числом q — порядком точки P;

кофактором c — натуральным числом таким, что произведение cq определяет порядок группы точек эллиптической кривой E.

Общая ключевая информация вырабатывается в ходе выполнения каждой сессии протокола и представляет собой точку Q, принадлежащую эллиптической кривой E и удовлетворяющую равенству

$$Q = kP,$$

где $k \equiv k_c k_s c \pmod q$, а величины k_c, $k_s \in F_q$ — случайные вычеты, вырабатываемые, соответственно, клиентом и сервером независимо друг от друга, см. 5.7.1 и 5.7.2. Требования к механизмам генерации случайных значений k_c, k_s определяются в Р 1323565.1.012—2017, см. раздел 5.

5.2.3 Ключи шифрования и ключи выработки имитовставки

В ходе выполнения протокола клиентом и сервером вырабатывается ключевая информация, предназначенная для шифрования и контроля целостности сообщений, передаваемых в ходе выполнения протокола выработки ключей:

а) ключевая информация SHTS $\in B^{64}$, предназначенная для зашифрования и контроля целостности сообщений, передаваемых от сервера к клиенту. Данная ключевая информация используется клиентом для расшифрования и проверки целостности получаемых от сервера сообщений. Ключевая информация SHTS позволяет определить следующие ключи:

- eSHTK $\in B^{32}$ — ключ, предназначенный для зашифрования сообщений, направляемых сервером клиенту. Данный ключ используется клиентом для расшифрования полученных от сервера сообщений;
- iSHTK $\in B^{32}$ — ключ, предназначенный для выработки имитовставки сообщений, направляемых сервером клиенту. Данный ключ используется клиентом для проверки целостности полученных от сервера сообщений.

Указанные ключи определяются равенствами:

$$eSHTK = SHTS[0 ,..., 31],\ iSHTK = SHTS[32, ..., 63];$$

б) ключевая информация CHTS $\in B^{64}$, предназначенная для зашифрования и контроля целостности сообщений, передаваемых от клиента к серверу. Данная ключевая информация используется сервером для расшифрования и проверки целостности получаемых от клиента сообщений.

Ключевая информация CHTS позволяет определить следующие ключи:

- eCHTK $\in B^{32}$ — ключ, предназначенный для зашифрования сообщений, направляемых клиентом серверу. Данный ключ используется сервером для расшифрования полученных от клиента сообщений;
- iCHTK $\in B^{32}$ — ключ, предназначенный для выработки имитовставки сообщений, направляемых клиентом серверу. Данный ключ используется сервером для проверки целостности полученных от клиента сообщений.

Указанные ключи определяются равенствами:

$$eCHTK = CHTS[0, ..., 31],\ iCHTK = CHTS[32, ..., 63]$$

Определенная выше ключевая информация вырабатывается клиентом и сервером в ходе выполнения протокола выработки ключей. Для выработки используется общая ключевая информация Q, ключи аутентификации iPSK и/или ePSK, а также сообщения, передаваемые в ходе выполнения протокола. Алгоритм выработки указанной ключевой информации описывается в 5.6.1.

5.2.4 Ключевая информация для протокола передачи прикладных данных

В результате выполнения протокола выработки ключей формируется следующая ключевая информация:

а) ключевая информация SATS $\in B^{64}$, предназначенная для выработки производных ключей, используемых для зашифрования и контроля целостности сообщений, передаваемых в ходе выполнения протокола передачи прикладных данных от сервера к клиенту. Ключевая информация используется клиентом для расшифрования и проверки целостности получаемых от сервера сообщений;

б) ключевая информация CATS $\in B^{64}$, предназначенная для выработки производных ключей, используемых для зашифрования и контроля целостности сообщений, передаваемых в ходе выполнения протокола передачи прикладных данных от клиента к серверу. Ключевая информация используется сервером для расшифрования и проверки целостности получаемых от клиента сообщений.

Определенная выше ключевая информация вырабатывается клиентом и сервером в ходе выполнения протокола выработки ключей. Для выработки ключей используется общая ключевая информация Q, ключи аутентификации iPSK и/или ePSK, а также сообщения, передаваемые в ходе выполнения протокола. Алгоритм выработки указанной ключевой информации описывается в 5.6.2.

5.3 Идентификация абонентов

Сервер и опционально клиент должны обладать собственными идентификаторами — последовательностями октетов произвольной длины, обозначаемыми, соответственно ID_s и ID_c.

Длины идентификаторов должны быть отличны от нуля и могут принимать произвольные натуральные значения. Рекомендуется определять идентификаторы последовательностями октетов длиной не менее восьми.

5.3.1 Определение идентификаторов

Идентификаторы клиента и сервера могут быть определены следующими способами:

а) при помощи сертификатов ключей проверки электронной подписи, обеспечивающих однозначное связывание идентификатора абонента с его ключом проверки электронной подписи; в этом случае в качестве идентификатора абонента выступает значение поля owner сертификата ключа проверки электронной подписи, представленное в ber-кодировке, см. Р 1323565.1.023—2018;

б) назначены на этапе производства контрольного и измерительного устройства либо в процессе штатной смены ключа аутентификации ePSK; рекомендуемый механизм связывания идентификаторов контрольных и измерительных устройств с предварительно распределенными ключами аутентификации описывается в приложении Б.

5.3.2 Обмен идентификаторами

Идентификаторы могут быть переданы от одного абонента к другому при помощи следующих механизмов:

а) путем направления расширения CertificateExtension, содержащего сертификат ключа проверки электронной подписи абонента, см. 5.7.2 и 5.7.3;

б) путем направления расширения RequestIdentifierExtension, содержащего идентификатор абонента, см. 5.8.5;

в) путем предварительного распределения сертификатов ключей проверки электронной подписи и/или идентификаторов абонентов на этапе производства контрольного и измерительного устройства, либо в процессе штатной смены ключа аутентификации ePSK; данные механизмы настоящими рекомендациями не регламентируются.

5.4 Аутентификация абонентов

Протокол выработки общих ключей позволяет обеспечить как одностороннюю, так и взаимную аутентификацию абонентов. В случае односторонней аутентификации клиент всегда выполняет аутентификацию сервера.

Аутентификация может быть реализована при помощи следующих криптографических механизмов:

а) механизма ключей проверки электронной подписи; данный механизм позволяет реализовать как одностороннюю, так и взаимную аутентификацию;

б) механизма предварительно распределенных ключей аутентификации; данный механизм позволяет реализовать только взаимную аутентификацию;

в) использования общего для клиента и сервера ключа аутентификации iPSK, см. 5.2.1, выработанного в ходе предыдущего сеанса защищенного взаимодействия; данный механизм аутентификации позволяет наследовать свойство односторонней или взаимной аутентификации из предыдущего сеанса защищенного взаимодействия.

Механизм аутентификации выбирается клиентом при инициализации протокола выработки общих ключей, см. 5.7.1.

5.4.1 Аутентификация с использованием ключей электронной подписи

В данном криптографическом механизме аутентификация сервера (клиента) производится путем проверки клиентом (сервером) истинности следующих утверждений:

а) сервер (клиент) обладает действительным сертификатом ключа проверки электронной подписи, позволяющим однозначно связать уникальный идентификатор сервера (клиента) с ключом проверки электронной подписи; алгоритм подтверждения того, что сертификат ключа проверки электронной подписи является действительным, описывается в 5.8.4;

б) сервер (клиент) обладает ключом электронной подписи, однозначно связанным с ключом проверки электронной подписи, для сертификата которого была проверена действительность.

Подтверждение того, что сервер (клиент) обладает ключом проверки электронной подписи проводится путем проверки истинности электронной подписи, выработанной сервером (клиентом) под случайным сообщением, формируемым клиентом и сервером в ходе выполнения протокола выработки ключей.

Сертификаты ключей проверки электронной подписи, используемые для аутентификации участников защищенного взаимодействия, могут распределяться следующим образом:

а) обмен сертификатами может производиться в ходе выполнения протокола выработки общих ключей;

б) путем предварительного распределения на этапе производства либо в процессе штатной эксплуатации контрольных и измерительных устройств; механизм предварительного распределения настоящими рекомендациями не регламентируются;

Допускается ситуация, в которой сертификаты, обмен которыми произошел в ходе одного сеанса защищенного взаимодействия, рассматриваются как предварительно распределенные и используются в последующих сеансах защищенного взаимодействия без обмена в ходе выполнения протокола выработки общих ключей.

В ходе выполнения протокола выработки ключей клиент (сервер) может:

а) запросить у сервера (клиента) сертификат ключа проверки электронной подписи, который будет использован для аутентификации сервера (клиента) (в запросе может быть указан конкретный удостоверяющий центр, выдавший сертификат ключа проверки электронной подписи); такой запрос должен быть направлен в случае, когда обмен сертификатами осуществляется в ходе выполнения протокола выработки общих ключей.

Примечание — Запрос сертификата должен производиться с помощью расширения RequestCertificateExtension в ходе первого или второго этапов протокола выработки общих ключей, см. 5.7.1 и см. 5.7.2. В ответ на запрос клиента (сервера) сервер (клиент) должен направить сертификат ключа проверки электронной подписи, используя для этого расширение CertificateExtension;

б) указать серверу (клиенту) сертификат ключа проверки электронной подписи, который должен использоваться для аутентификации сервера (клиента), при этом может быть указан как номер сертификата ключа проверки электронной подписи, так и идентификатор удостоверяющего центра, выдавшего сертификат ключа проверки электронной подписи; такое указание должно направляться в случае, когда осуществлен предварительный обмен сертификатами ключей проверки электронной подписи.

Примечание — Указание об использовании сертификата должно направляться с помощью расширения SetCertificateExtension в ходе первого или второго этапов протокола выработки общих ключей, см. 5.7.1 и 5.7.2. В случае указания об использовании сертификата с заданным удостоверяющим центром сервер (клиент) должен направить клиенту (серверу) сертификат ключа проверки электронной подписи, используя для этого расширение CertificateExtension;

в) указать серверу (клиенту) номер сертификата ключа проверки электронной подписи, который должен использоваться для собственной аутентификации клиента (сервера); такое указание должно направляться в случае, когда осуществлен предварительный обмен сертификатами ключей проверки электронной подписи.

Примечание — Указание номера используемого сертификата должно направляться с помощью расширения InformCertificateExtension в ходе первого или второго этапов протокола выработки общих ключей, см. 5.7.1 и 5.7.2. Направление данного расширения не подразумевает ответа от сервера (клиента).

В случае, если клиентом выбран механизм аутентификации с использованием сертификатов ключей проверки электронной подписи и не направлен запрос или указание на использование заданного сертификата ключа проверки электронной подписи, сервер в обязательном порядке должен отправить клиенту расширение CertificateExtension, содержащее сертификат ключа проверки электронной подписи.

5.4.2 Аутентификация с использованием предварительно распределенных ключей

В данном криптографическом механизме выполняется взаимная аутентификация клиента и сервера. Аутентификация производится путем проверки истинности следующего утверждения:

а) сервер (клиент) обладает секретным предварительно распределенным ключом аутентификации, значение которого совпадает со значением предварительно распределенного ключа аутентификации, которым обладает клиент (сервер);

В качестве предварительно распределенного ключа аутентификации может выступать ключ ePSK, либо ключ iPSK, выработанный в ходе предыдущего сеанса защищенного взаимодействия, см. 5.2.1.

Подтверждение того, что сервер (клиент) обладает предварительно распределенным ключом аутентификации проводится клиентом (сервером) путем проверки совпадения значения кода целостности случайного сообщения, выработанного совместно клиентом и сервером в ходе выполнения протокола выработки общих ключей, со значением, полученным от сервера (клиента), в зашифрованном виде.

5.5 Формирование последовательностей октетов

Для выработки ключей шифрования и ключей выработки имитовставки, используемых в ходе выполнения протокола выработки ключей, а также ключевой информации для протокола передачи прикладных данных, используются последовательности октетов R_1, R_2 и H_1, H_2, H_3, H_4 и H_5.

Формирование последовательностей октетов R_1 и R_2 происходит в соответствии со следующими правилами:

а) последовательность октетов R_1 полагается равной Ser(x(Q)); способ преобразования определен в В.3.10;

б) последовательность октетов R_2 полагается равной ID_s, где ID_s — идентификатор сервера;

в) если сервером после формировании сообщения ServerHelloMessage был запрошен сертификат ключа проверки подписи клиента $Cert_c$ или идентификатор клиента ID_c, см. 5.7.2, то текущее значение последовательности октетов R_2 конкатенируется со значением ID_c, т. е.

$$R_2 = R_2||ID_c;$$

г) если при создании сообщения ClientHelloMessage клиентом был использован идентификатор ID_{iPSK} ключа аутентификации iPSK, см. 5.7.1, то текущее значение последовательности октетов R_1 и текущее значение последовательности октетов R_2 конкатенируются со значением ключа iPSK, т. е.

$$R_1 = R_1||iPSK, R_2 = R_2||iPSK;$$

д) если при создании сообщения ClientHelloMessage клиентом был использован идентификатор ID_{ePSK} ключа аутентификации ePSK, см. 5.7.1, то текущее значение последовательности октетов R_1 и текущее значение последовательности октетов R_2 конкатенируются со значением ключа ePSK, т. е.

$$R_1 = R_1||ePSK, R_2 = R_2||ePSK.$$

Примечание — В краткой форме последовательности октетов R_1 и R_2 могут быть записаны в виде

$$R_1 = Ser(x(Q))||iPSK^*||ePSK^*, R_2 = ID_s||ID_c||iPSK^*||ePSK^*,$$

где знак «*» означает, что данная последовательность октетов является опциональной и ее включение в состав последовательностей октетов R_1 и/или R_2 зависит от действий клиента и сервера при создании сообщений ClientHelloMessage и ServerHelloMessage.

Формирование последовательностей октетов H_1, H_2, H_3, H_4 и H_5 происходит по следующим правилам.

1 Последовательность октетов H_1 формируется следующим образом

- последовательность октетов H_1 полагается равной сообщению ClientHelloMessage;
- если клиентом после отправки сообщения ClientHelloMessage отправлялись расширения ExtensionMessage1, ... , ExtensionMessageNc, то текущее значение последовательности октетов H_1 конкатенируется с указанными сообщениями в порядке их отправления, т. е.

$$H_1 = H_1||ExtensionMessage1||...||ExtensionMessageNc$$

- текущее значение последовательности октетов H_1 объединяется с сообщением ServerHelloMessage, то есть $H_1 = H_1||ServerHelloMessage$.

Примечание — Последовательность октетов H_1 представляет собой последовательную конкатенацию всех переданных в открытом виде сообщений. Правильный порядок отправки расширений может быть определен с помощью уникальных номеров фреймов, см. 7.3.

2 Последовательность октетов H_2 формируется следующим образом

- последовательность октетов H_2 определяется равенством $H_2 = H_1$;
- если сервером после сообщения ClientHelloMessage отправлялись расширения ExtensionMessage1, ... , ExtensionMessageNs, то текущее значение последовательности октетов H_2 конкатенируется с указанными расширениями в незашифрованном виде, в порядке их отправления, т. е.

$$H_2 = H_2||ExtensionMessage1||...||ExtensionMessageNs$$

3 Последовательность октетов H_3 формируется следующим образом

- последовательность октетов H_3 определяется равенством $H_3 = H_2$;
- текущее значение последовательности октетов H_3 конкатенируется с отправленным сервером клиенту сообщением VerifyMessage в незашифрованном (расшифрованном) виде, т. е.

$$H_3 = H_3||VerifyMessage.$$

4 Последовательность октетов H_4 формируется следующим образом

- последовательность октетов H_4 определяется равенством $H_4 = H_3$;
- если клиентом в ответ на сообщение ServerHelloMessage отправлялись расширения ExtensionMessageC1, ... , ExtensionMessageCc, то текущее значение последовательности октетов H_4 конкатенируется с указанными расширениями в незашифрованном виде, в порядке их отправления, т. е.

$$H_4 = H_4||ExtensionMessageC1||...||ExtensionMessageCc$$

5 Последовательность октетов H_5 формируется следующим образом

- последовательность октетов H_5 определяется равенством $H_5 = H_4$;
- текущее значение последовательности октетов H_5 конкатенируется с отправленным клиентом серверу сообщением VerifyMessage в незашифрованном (расшифрованном) виде, т. е.

$$H_5 = H_5||\text{VerifyMessage}.$$

Примечания

1 Последовательности октетов H_1, .., H_5 представляют собой конкатенации сообщений и расширений, последовательно передаваемых в ходе выполнения протокола выработки ключей. Схема процесса выработки последовательностей октетов H_1, .., H_5 изображена на рисунке 3.

2 Поскольку отправка расширений является опциональной, то последовательности октетов H_1 и H_2, а также H_3 и H_4, могут совпадать.

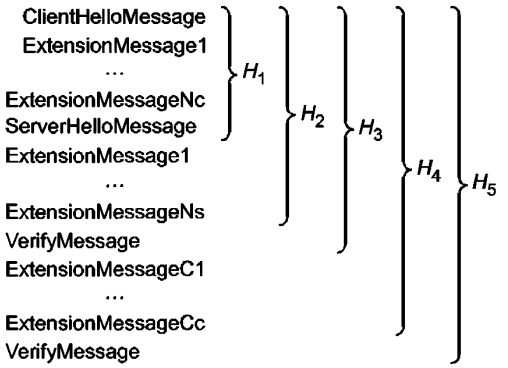

Рисунок 3 — Схема формирования последовательностей октетов

5.6 Формирование ключевой информации

В этом подразделе описываются алгоритмы формирования ключевой информации, определенной в 5.2.

5.6.1 Формирование ключей шифрования и ключей выработки имитовставки

Ключи шифрования eSHTK, eCHTK и ключи выработки имитовставки iSHTK, iCHTK вырабатываются клиентом и сервером в ходе выполнения протокола выработки ключей.

Указанные ключи используются для шифрования информации, передаваемой только в ходе выполнения протокола выработки ключей.

При формировании ключей eSHTK, eCHTK и iSHTK, iCHTK используется общая ключевая информация Q, а также определенные в 5.5 последовательности октетов R_1 и H_1, H_3.

Для формирования ключей необходимо выполнить последовательность действий:

а) вычислить последовательность октетов $K \in B^{64}$, определяемую равенством

$$K = \text{Streebog}_{512}(R_1);$$

б) вычислить последовательность октетов $SHTS \in B^{64}$, удовлетворяющую равенству

$$SHTS = \text{HMAC}_{512}(K, \text{Streebog}_{512}(H_1)),$$

и определить ключи eSHTS и iSHTS равенствами

$$eSHTK = SHTS[0, \ldots, 31],\ iSHTK = SHTS[32, \ldots, 63];$$

в) вычислить последовательность октетов $CHTS \in B^{64}$, удовлетворяющую равенству

$$CHTS = \text{HMAC}_{512}(K, \text{Streebog}_{512}(H_3));$$

и определить ключи eCHTS и iCHTS равенствами

$$eCHTK = CHTS[0, \ldots, 31],\ iCHTK = CHTS[32, \ldots, 63].$$

Описанная выше последовательность действий схематично изображена на рисунке 4.

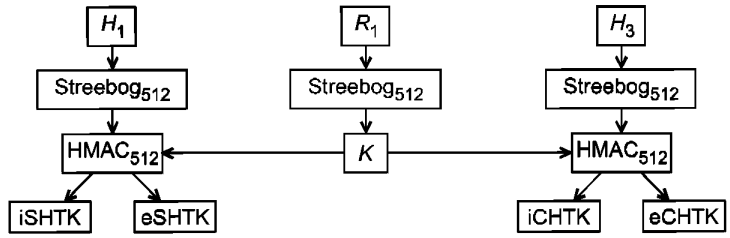

Рисунок 4 — Схема выработки ключей для шифрования сообщений в ходе выполнения протокола формирования общей ключевой информации

5.6.2 Формирование ключей шифрования и ключей выработки имитовставки протокола передачи прикладных данных

Ключевая информация CATS, SATS является целью выполнения протокола выработки ключей. При формировании указанной ключевой информации используется алгоритм PRF_TLS_GOSTR3411_2012_512, регламентируемый Р 50.1.113—2016, а также общая ключевая информация Q и последовательности октетов R_2, H_5, определенные в 5.5.

Для формирования ключевой информации CATS, SATS необходимо выполнить последовательность действий:

а) вычислить последовательность октетов $T \in B^{64}$, определяемую равенством

$$T = HMAC_{512}(Ser(x(Q)), R_2);$$

б) вычислить последовательность октетов $A_0 \in B^{64}$, определяемую равенством

$$A_0 = Streebog_{512}(H_5);$$

в) вычислить последовательность октетов $A_1 \in B^{64}$, определяемую равенством

$$A_1 = HMAC_{512}(T, A_0);$$

г) определить последовательность октетов $CATS \in B^{64}$равенством

$$CATS = HMAC_{512}(T, A_1||A_0),$$

в котором операция сжатия применяется последовательно сначала к последовательности октетов A_1, а потом к последовательности A_0;

д) вычислить последовательность октетов $A_2 \in B^{64}$, определяемую равенством

$$A_2 = HMAC_{512}(T, A_1);$$

е) определить последовательность октетов $SATS \in B^{64}$ равенством

$$SATS = HMAC_{512}(T, A_2||A_0),$$

в котором операция сжатия применяется последовательно сначала к последовательности октетов A_1, а потом к последовательности A_0.

Описанная выше последовательность действий схематично изображена на рисунке 5.

5.7 Описание протокола

В этом подразделе дается детальное описание последовательности действий, производимых клиентом и сервером в ходе выполнения протокола выработки ключей.

Действия, описываемые в 5.7.1 и 5.7.3, выполняются клиентом; действия, описываемые в 5.7.2 и 5.7.4, выполняются сервером.

5.7.1 Формирование сообщения ClientHelloMessage

На первом этапе протокола выработки ключей клиентом выполняется процедура инициализации, в ходе которой клиент формирует сообщение ClientHelloMessage, см. В.5.1, а также, при необходимости, дополнительные расширения.

Для формирования сообщения ClientHelloMessage клиент выполняет последовательность действий:

а) клиент вырабатывает случайную последовательность октетов длиной 32 октета и помещает ее в поле ClientHelloMessage.random;

б) клиент выбирает идентификатор ID_E одной из эллиптических кривых, определяемых типом данных EllipticCurveID, см. В.3.9;

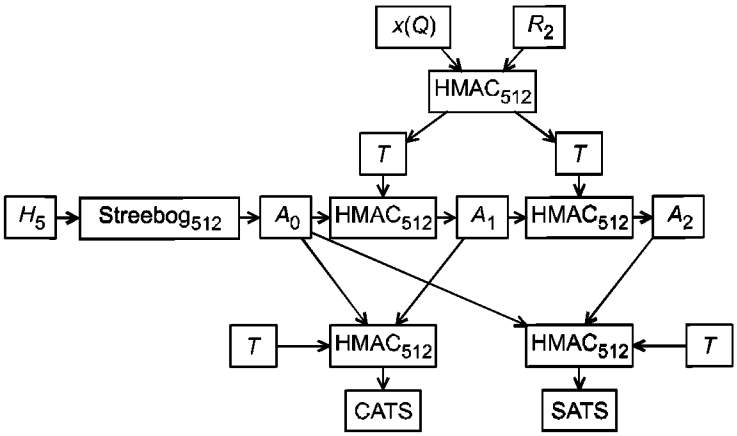

Рисунок 5 — Схема выработки ключей для протокола передачи прикладных данных

в) клиент вырабатывает случайное целое число k_c, удовлетворяющее неравенствам $0 < k_c < q$, где q порядок точки P, определяемой параметрами эллиптической кривой, выбранной клиентом на предыдущем шаге.

Примечания

1 Вырабатываемое клиентом значение k_c является криптографически опасной информацией, знание которой может привести к определению вырабатываемой ключевой информации и нарушению конфиденциальности передаваемой информации. В связи с этим рекомендуется удалять значение k_c на следующем шаге, сразу после использования.

2 Использование одного и того же генератора случайных чисел для выработки значения k_c и случайной последовательности октетов длиной 32 октета может быть потенциально опасным — нарушитель может использовать передаваемые в открытом виде элементы последовательности для определения секретного значения k_c. В связи с этим рекомендуется использование различных генераторов случайных чисел для генерации значения k_c и случайной последовательности октетов длиной 32 октета. В случае невозможности применения различных генераторов, рекомендуется использовать генератор, удовлетворяющий Р 1323565.1.012—2017;

г) клиент вычисляет точку кривой $P_c = k_cP$ и помещает координаты вычисленной точки в поля структуры ClientHelloMessage.EllipticCurvePoint, т. е.

$$ClientHelloMessage.point.id = ID_E;$$

$$ClientHelloMessage.point.x = Ser(x(P_c));$$

$$ClientHelloMessage.point.y = Ser(y(P_c));$$

д) если клиент хочет использовать для аутентификации сервера предварительно распределенный ключ аутентификации ePSK, то клиент определяет:

$$ClientHelloMessage.idpsk.present = isPresent;$$

$$ClientHelloMessage.idpsk.type = ePSKKey;$$

$$ClientHelloMessage.idpsk.length = Len(ID_{ePSK});$$

$$ClientHelloMessage.idpsk.id = ID_{ePSK},$$

где ID_{ePSK} — идентификатор ключа ePSK, а значения isPresent и notPresent определены в В.3.2, В противном случае клиент определяет

$$ClientHelloMessage.idpsk.present = notPresent.$$

Примечания

1 В случае аутентификации по ключу ePSK аутентификация сервера с использованием ключа аутентификации iPSK или сертификата ключа проверки электронной подписи, см. 5.2.1, может не проводиться.

2 Если клиент хочет выполнить аутентификацию сервера как с использованием ключа аутентификации ePSK, так и с использованием сертификата ключа проверки подписи, то после отправки сообщения ClientHelloMessage клиент должен сформировать и направить серверу расширение RequestCertificateExtension;

е) если клиент хочет использовать для аутентификации сервера вычисленный в ходе выполнения предыдущего сеанса защищенного взаимодействия ключ аутентификации iPSK, то клиент определяет:

ClientHelloMessage.idpsk.present = isPresent;

ClientHelloMessage.idpsk.type = iPSKKey;

ClientHelloMessage.idpsk.length = Len(ID_{iPSK});

ClientHelloMessage.idpsk.id = ID_{iPSK},

где ID_{iPSK} — идентификатор ключа iPSK, а значения isPresent и notPresent определены в В.3.2. В противном случае клиент определяет:

ClientHelloMessage.idpsk.present = notPresent.

Примечания

1 Использование ключа аутентификации iPSK допускается только после его выработки в ходе предыдущего сеанса защищенного взаимодействия в соответствии с 6.6.

2 В случае аутентификации по ключу iPSK аутентификация сервера с использованием ключа аутентификации ePSK или сертификата ключа проверки электронной подписи, см. 5.2.1, может не проводиться.

3 Если клиент хочет выполнить аутентификацию сервера как с использованием ключа аутентификации iPSK, так и с использованием сертификата ключа проверки подписи, то после отправки сообщения ClientHelloMessage клиент должен сформировать и направить серверу расширение RequestCertificateExtension.

4 Если клиент не указал ни одного идентификатора предварительно распределенного ключа, то клиент обязан выполнить аутентификацию сервера с использованием сертификата ключа проверки подписи;

ж) клиент определяет количество расширений N_c, которые будут направлены им серверу, и помещает это число в поле ClientHelloMessage.countOfExtensions;

з) клиент выбирает криптографический механизм контроля целостности сообщений и расширений, которые будут передаваться в незашифрованном виде; идентификатор выбранного криптографического механизма должен определяться перечислением CryptoMechanism и помещаться клиентом в поле ClientHelloMessage.algorithm.

Примечания

1 В большинстве случае для контроля целостности рекомендуется использовать алгоритм бесключевого хеширования.

2 В случае, когда для аутентификации сервера используются ключи iPSK или ePSK, для контроля целостности передаваемых в незашифрованном виде сообщений и расширений клиентом может быть выбрать алгоритм выработки имитовставки. При этом, для выработки имитовставки используется ключ аутентификации iPSK или ePSK.

3 Клиент может поместить в поле ClientHelloMessage.algorithm значение идентификатора с указанием параметров криптографических механизмов, используемых для обеспечения конфиденциальности и целостности сообщений и расширений, передаваемых в зашифрованном виде; данные значения могут быть учтены сервером при выборе значения идентификатора криптографических механизмов, выбираемого сервером в ходе формирования сообщения ServerHelloMessage, см. 5.7.2;

и) в соответствии с описанным в В.5.1 методом созданное сообщение ClientHelloMessage представляется в виде последовательности октетов и передается транспортному протоколу для отправки серверу в незашифрованном виде.

В случае если клиентом принято решение об отправке серверу расширений, см. перечисление ж), то клиент переходит к их формированию. Допускается использование клиентом следующих расширений:

RequestCertificateExtension или SetCertificateExtension;

RequestIdentifierExtension;

KeyMechanismExtension.

Для обеспечения целостности формируемых клиентом расширений используется алгоритм контроля целостности, установленный клиентом в поле ClientHelloMessage.algorithm. После формирования расширения передаются транспортному протоколу для отправки серверу в незашированном виде.

Примечание — Выбранный клиентом способ аутентификации сервера влияет на содержимое обязательного сообщения VerifyMessage, отправляемого сервером клиенту, см. 5.7.2. В случае, когда клиентом выбрана аутентификация по предварительно распределенному ключу iPSK или ePSK, отправляемое сервером сообщение VerifyMessage всегда содержит код целостности и выполнено

VerifyMessage.mac.present = isPresent.

В случае, когда идентификаторы предварительно распределенных ключей в сообщении ClientHelloMessage не указаны, либо клиентом добавлено расширение RequestCertificateExtension, отправляемое сервером сообщение VerifyMessage содержит значение электронной подписи и

VerifyMessage.sign.present = isPresent.

5.7.2 Формирование сообщения ServerHelloMessage

На втором этапе выполнения протокола выработки общих ключей сервер получает отправленные клиентом сообщения и, при наличии, расширения, выполняет контроль целостности полученных данных, формирует сообщение ServerHelloMessage, а также, при необходимости, формирует собственные расширения.

При получении от клиента сообщения ClientHelloMessage сервер выполняет следующие проверки:

а) сервер проверяет, что значение поля ClientHelloMessage.algorithm содержит константу, определяемую типом данных CryptoMechanism. Если это не так, то сервер отправляет клиенту сообщение AlertMessage со значением unsupportedCryptoMechanism и завершает сеанс защищенного взаимодействия.

Примечание — Сообщение AlertMessage со значением ошибки unsupportedCryptoMechanism может отправляться клиенту также в том случае, когда сервер не поддерживает реализацию указанного алгоритма;

б) если значение поля ClientHelloMessage.idpsk.present содержит константу isPresent, то выполняются следующие проверки:

1) сервер определяет тип переданного идентификатора ключа аутентификации, содержащийся в поле ClientHelloMessage.idpsk.type;

2) если идентификатор ключа, содержащийся в поле ClientHelloMessage.idpsk.id, отвергается сервером, то сервер отправляет клиенту сообщение AlertMessage со значением wrongPreSharedKey и завершает сеанс защищенного взаимодействия;

Примечание — Причины, по которым сервер может отвергнуть ключ аутентификации ePSK или iPSK настоящими рекомендациями не регулируются. В качестве примеров таких причин могут выступать: неизвестное значение идентификатора ключа, исчерпание временного интервала использования ключа, превышение числа допустимых использований ключа, компрометация ключа и т. п.;

3) если значение поля ClientHelloMessage.algorithm содержит в себе значение алгоритма выработки имитовставки, то сервер выполняет контроль целостности сообщения ClientHelloMessage с использованием указанного алгоритма и ключа аутентификации ePSK или iPSK; в случае нарушения целостности сервер отправляет клиенту сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия; в случае успешной проверки целостности сервер переходит к перечислению г);

в) сервер выполняет контроль целостности сообщения ClientHelloMessage с использованием алгоритма бесключевого хеширования; в случае нарушения целостности сервер отправляет клиенту сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия;

г) сервер проверяет корректность идентификатора эллиптической кривой, содержащегося в поле ClientMessageHello.point.id; если значение данного идентификатора не является допустимым, то сервер отправляет клиенту сообщение AlertMessage со значением unsupportedEllipticCurveID и завершает сеанс защищенного взаимодействия;

д) используя алгоритм из 5.9.1, сервер проверяет корректность точки эллиптической кривой P_c, содержащейся в структуре ClientHelloMessage.point; в случае нарушения корректности точки эллиптической кривой сервер отправляет клиенту сообщение AlertMessage со значением wrongEllipticCurvePoint и завершает сеанс защищенного взаимодействия;

е) в случае если поле ClientHelloMessage.countOfExtensions содержит отличное от нуля значение, то сервер переходит к получению из канала связи и проверке целостности полученных от клиента расширений, используя при этом алгоритм, определенный значением поля ClientHelloMessage.algorithm. В случае, если целостность хотя бы одного расширения нарушена, сервер отправляет клиенту сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия;

ж) для каждого из полученных расширений сервер проверяет корректность содержащегося в расширении запроса; в случае неверного запроса или отсутствии возможности удовлетворить запрос клиента, сервер отправляет клиенту сообщение AlertMessage с информацией об ошибке и завершает сеанс защищенного взаимодействия. Коды возможных ошибок приводятся в 5.8.

Примечание — Процедура контроля целостности сообщения ClientHelloMessage и последующих за ним расширений может быть реализована на уровне транспортного протокола, см. 7.5.1.

После успешного прохождения всех перечисленных выше проверок сервер формирует сообщение ServerHelloMessage и, в случае необходимости, расширения, см. 5.8.

Для этого сервер выполняет следующие действия:

з) сервер вырабатывает случайную последовательность октетов длиной 32 октета и помещает ее в поле ServerHelloMessage.random;

и) сервер вырабатывает случайное целое число k_s, удовлетворяющее неравенствам $0 < k_s < q$, где q порядок точки P, определяемой параметрами эллиптической кривой, идентификатор которой выбран клиентом и содержится в поле ClientHelloMessage.point.id.

Примечание — Высказанные ранее при формировании сообщения ClientHelloMessage положения, см. 5.7.1, примечание к перечислению в), о необходимости использования различных генераторов случайных чисел для выработки значения k_s и случайной последовательности октетов длиной 32 октета, также должны применяться при формировании сообщения ServerHelloMessage;

к) сервер вычисляет точку кривой $P_s = k_sP$ и помещает координаты вычисленной точки в поля структуры ServerHelloMessage.EllipticCurvePoint, т. е.:

$$\text{ServerHelloMessage.point.id} = \text{ClientHelloMessage.point.id},$$

$$\text{ServerHelloMessage.point.x} = \text{Ser}(x(P_S)),$$

$$\text{ServerHelloMessage.point.y} = \text{Ser}(y(P_S));$$

л) сервер определяет количество расширений N_s, которые будут направлены им клиенту, и помещает это число в поле ServerHelloMessage.countOfExtensions;

м) сервер выбирает криптографические механизмы, используемые для шифрования и контроля целостности передаваемых далее сообщений и расширений; идентификатор выбранных криптографических механизмов помещается сервером в поле ServerHelloMessage.algorithm.

Примечания

1 Решение о выборе криптографических механизмов, используемых для шифрования и контроля целостности передаваемых сообщений, принимается сервером. При выборе механизмов сервер может учесть «пожелания» клиента, содержащиеся в поле ClientHelloMessage.algorithm, однако такое поведение сервера не является обязательным.

2 Выбранные сервером в перечислении м) криптографические механизмы используются далее для передачи зашифрованных сообщений как в протоколе выработки ключей, так и в протоколе передачи прикладных данных;

н) сервер передает сформированное сообщение ServerHelloMessage транспортному протоколу для отправки клиенту в незашифрованном виде. Для контроля целостности сообщения ServerHelloMessage используется алгоритм, указанный в поле ClientHelloMessage.algorithm;

о) согласно 5.2.2 сервер вырабатывает общую ключевую информацию Q, определяемую равенством $Q = ck_sP_c$, где значение кофактора c определяется параметрами используемой эллиптической кривой;

п) сервер вырабатывает ключевую информацию SHTS — для этого сервер формирует определенную в 5.5 последовательность октетов H_1 и применяет определенный в 5.6.1 алгоритм формирования ключевой информации;

р) если поле idpsk.present сообщения ClientHelloMessage не содержит значение isPresent или клиентом направлено одно из расширений RequestCertificateExtension или SetCertificateExtension, то сервер выполняет следующие действия:

1) сервер выбирает ключ проверки электронной подписи Q_s, который будет использован клиентом для аутентификации сервера; если клиентом направлено расширение RequestCertificateExtension или SetCertificateExtension, то выбор ключа Q_s проводится с учетом значений, помещенных клиентом в соответствующее расширение. В случае отсутствия у сервера ключ проверки электронной подписи Q_s, удовлетворяющего запрашиваемым клиентом требованиям, то сервер отправляет клиенту сообщение AlertMessage с информацией об ошибке, см. В.3.15, и завершает сеанс защищенного взаимодействия.

Примечание — Факт того, что значения, указанные клиентом в расширении, являются корректными, проверен сервером ранее в перечислении ж);

2) за исключением случая, когда выполнено равенство SetCertificateExtension.certproctype = number, сервер формирует:

либо расширение CertificateExtension, содержащее сертификат ключа проверки электронной подписи Q_s;

либо расширение InformCertificateExtension, содержащее номер предварительно распределенного сертификата ключа проверки электронной подписи Q_s;

после этого сервер передает сформированное расширение транспортному протоколу для отправки клиенту в зашифрованном виде; для шифрования и контроля целостности сформированного расширения используются криптографические механизмы, указанные в поле ServerHelloMessage.algorithm, и выработанная ранее ключевая информация SHTS.

Примечание — Случай равенства SetCertificateExtension.certproctype = number означает, что клиент обладает сертификатом ключа проверки электронной подписи сервера и в явном виде указывает его номер. В этом случае передача сертификата от сервера к клиенту является избыточной;

с) если сервер хочет аутентифицировать клиента с помощью механизма электронной подписи, сервер формирует расширение RequestCertificateExtension или SetCertificateExtension, содержащее запрос сертификата ключа проверки электронной подписи клиента и передает сформированное расширение транспортному протоколу для отправки клиенту в зашифрованном виде; для шифрования и контроля целостности передаваемого расширения используются криптографические механизмы, указанные в поле ServerHelloMessage.algorithm, и выработанная ранее ключевая информация SHTS;

т) в случае необходимости, в соответствии с 5.8, сервером могут быть сформированы расширения:

RequestIdentifierExtension;

KeyMechanismExtension.

Данные расширения передаются транспортному протоколу для отправки клиенту в зашифрованном виде; для шифрования и контроля целостности передаваемых расширений используются криптографические механизмы, указанные в поле ServerHelloMessage.algorithm, и выработанная ранее ключевая информация SHTS;

у) сервер формирует сообщение VerifyMessage. Для этого он выполняет следующие действия:

1) если ранее, см. перечисление р), сервером определен ключ проверки электронной подписи Q_s, то сервер формирует электронную подпись под последовательностью октетов H_2, см. 5.5, и помещает электронную подпись в сообщение VerifyMessage, т. е.:

$$\text{VerifyMessage.sign.present} = \text{isPresent};$$

$$\text{VerifyMessage.sign.length} = \text{len};$$

$$\text{VerifyMessage.sign.code} = \text{Sign}(d_s, \text{Streebog}_{512}(H_2)),$$

где d_s — ключ электронной подписи сервера, соответствующий ключу проверки подписи Q_s, а len — размер электронной подписи, определяемый ключом проверки подписи Q_s. В противном случае сервер определяет значение

$$\text{VerifyMessage.sign.present} = \text{notPresent};$$

2) если клиентом в составе сообщения ClientHelloMessage был использован идентификатор предварительно распределенного ключа iPSK или ePSK, то сервер формирует код целостности под последовательностью октетов H_2, см. 5.5, и помещает код целостности в сообщение VerifyMessage, т. е.:

$$\text{VerifyMessage.mac.present} = \text{isPresent};$$

$$\text{VerifyMessage.mac.length} = 16;$$

$$\text{VerifyMessage.mac.code} = \text{Streebog}_{512}(H_2)[0, \ldots, 15].$$

В противном случае сервер определяет значение

$$\text{VerifyMessage.mac.present} = \text{notPresent};$$

ф) после формирования сообщения VerifyMessage сервер передает его транспортному протоколу для отправки клиенту в зашифрованном виде; для шифрования и контроля целостности передаваемого сообщения VerifyMessage используются криптографические механизмы, указанные в поле ServerHelloMessage.algorithm, и выработанная ранее ключевая информация SHTS.

5.7.3 Завершающие действия клиента

На третьем этапе выполнения протокола выработки ключей клиент завершает аутентификацию сервера и формирует расширения, необходимые для собственной аутентификации и подтверждения выработанной ключевой информации.

После получения ответа от сервера клиент выполняет следующие проверки:

а) в случае получения сообщения AlertMessage клиент завершает сеанс защищенного взаимодействия; в противном случае клиент переходит к анализу полученного сообщения ServerHelloMessage;

б) клиент проверяет, что значение поля ServerHelloMessage.algorithm содержит константу, определяемую типом данных CryptoMechanism. Если это не так, то клиент отправляет серверу сообщение AlertMessage со значением ошибки unsupportedCryptoMechanism и завершает сеанс защищенного взаимодействия.

Примечание — Сообщение AlertMessage со значением ошибки unsupportedCryptoMechanism может отправляться клиентом также в случае, когда клиент не поддерживает реализацию указанного алгоритма;

в) клиент выполняет контроль целостности сообщения ServerHelloMessage с использованием алгоритма, определенного в поле ClientHelloMessage.algorithm. В случае нарушения целостности клиент отправляет серверу сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия;

г) используя алгоритм из 5.9.1, клиент проверяет корректность точки эллиптической кривой P_s, содержащейся в структуре ServerHelloMessage.point; в случае нарушения корректности точки эллиптической кривой клиент отправляет серверу сообщение AlertMessage со значением wrongEllipticCurvePoint и завершает сеанс защищенного взаимодействия;

д) согласно 5.2.2 клиент вырабатывает общую ключевую информацию Q — точку кривой, определяемую равенством $Q = ck_cP_s$, где значение кофактора c определяется парметрами используемой эллиптической кривой;

е) клиент вырабатывает ключевую информацию SHTS — для этого клиент формирует определенную в 5.5 последовательность октетов H_1 и применяет определенный в 5.6.1 алгоритм формирования ключевой информации;

ж) используя алгоритм, указанный в поле ServerHelloMessage.algorithm, клиент проверяет целостность и расшифровывает, при наличии, полученные от сервера расширения, а также сообщение VerifyMessage. Если при расшифровании указанных сообщений клиент определяет нарушение целостности хотя бы одного из полученных расширений и/или сообщений, клиент отправляет серверу сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия.

Примечание — Процедура контроля целостности сообщения ServerHelloMessage и последующих за ним расширений может быть реализована на уровне транспортного протокола, см. 7.5.1;

з) в случае, если поле ServerHelloMessage.countOfExtensions содержит отличное от нуля значение, то для каждого из полученных от сервера расширений клиент проверяет корректность содержащегося в расширении запроса; в случае неверного запроса или отсутствии возможности удовлетворить запрос сервера, клиент отправляет серверу сообщение AlertMessage с информацией об ошибке и завершает выполнение протокола; коды возможных ошибок приводятся в 5.8;

и) если клиентом ранее направлялось расширение RequestCertificateExtension или SetCertificateExtension с указанием конкретного сертификата ключа проверки электронной подписи, или клиентом было получено от сервера корректное расширение CertificateExtension, содержащее сертификат ключа проверки электронной подписи, или корректное расширение InformCertificateExtension, содержащее номер сертификата ключа электронной подписи, то клиент:

1) выбирает соответствующий сертификату ключ проверки электронной подписи Q_s;

2) проверяет, что поле VerifyMessage.sign.present принимает значение isPresent; если это не так, то клиент отправляет серверу сообщение AlertMessage со значением ошибки lostIntegrityCode и завершает сеанс защищенного взаимодействия;

3) с использованием ключа проверки электронной подписи Q_s проверяет истинность подписи под последовательностью октетов H2, см. 5.5, содержащейся в VerifyMessage.sign.code, т. е. выполнение тождества:

$$Verify(Q_{-s}, Streebog_{-512}(H_{-2}), VerifyMessage.sign.code) = True;$$

если тождество не выполнено, то клиент отправляет серверу сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия;

к) если клиентом в составе сообщения ClientHelloMessage был использован идентификатор предварительно распределенного ключа iPSK или ePSK, то клиент:

1) проверяет, что поле VerifyMessage.mac.present принимает значение isPresent; если это не так, то клиент отправляет серверу сообщение AlertMessage со значением ошибки lostIntegrityCode и завершает сеанс защищенного взаимодействия;

2) формирует код целостности под последовательностью октетов H_2, см. 5.5, и проверяет выполнимость равенства

$$VerifyMessage.mac.code = Streebog_{512}(H_2)[0, \ldots, 15];$$

если равенство не выполнено, то клиент отправляет серверу сообщение AlertMessage со значением ошибки wrongIntegrityCode и завершает сеанс защищенного взаимодействия.

После успешного выполнения всех перечисленных выше проверок клиент принимает решение об успешной аутентификации сервера и переходит к формированию и отправке серверу сообщений и, при необходимости, расширений. Для этого клиент выполняет следующие действия:

к) клиент вырабатывает ключевую информацию CHTS — для этого клиент формирует определенную в 5.5 последовательность октетов H_3 и применяет определенный в 5.6.1 алгоритм формирования ключевой информации;

л) если сервером направлено одно из расширений RequestCertificateExtension или SetCertificateExtension, то клиент:

1) выбирает ключ проверки электронной подписи Q_c, который будет использован сервером для аутентификации клиента; выбор ключа Q_c производится с учетом значений, помещенных сервером в соответствующее расширение;

2) за исключением случая, когда выполнено равенство SetCertificateExtension.certproctype = number, клиент формирует:

либо расширение CertificateExtension, содержащее сертификат ключа проверки электронной подписи Q_c;

либо расширение InformCertificateExtension, содержащее номер предварительно распределенного сертификата ключа проверки электронной подписи Q_c;

после этого клиент передает сформированное расширение транспортному протоколу для отправки серверу в зашифрованном виде; для шифрования и контроля целостности сформированного расширения используются криптографические механизмы, указанные в поле ServerHelloMessage.algorithm, и выработанная ранее ключевая информация CHTS.

Примечание — Случай равенства SetCertificateExtension.certproctype = number означает, что сервер обладает сертификатом ключа проверки электронной подписи клиента и в явном виде указывает его номер. В этом случае передача сертификата от клиента к серверу является избыточной;

м) клиент формирует сообщение VerifyMessage. Для этого он выполняет следующие действия:

1) если на предыдущем шаге клиентом был определен сертификат ключа проверки электронной подписи Q_c, то клиент формирует электронную подпись под последовательностью октетов H_4, см. 5.5, и помещает электронную подпись в сообщение VerifyMessage, т. е.:

$$\text{VerifyMessage.sign.present} = \text{isPresent};$$

$$\text{VerifyMessage.sign.length} = \text{len};$$

$$\text{VerifyMessage.sign.code} = \text{Sign}(d_c, \text{Streebog}_{512}(H_4)),$$

где d_c — ключ электронной подписи клиента, соответствующий ключу проверки подписи Q_c, а len — размер электронной подписи, определяемый ключом проверки подписи Q_c.

В противном случае клиент определяет значение

$$\text{VerifyMessage.sign.present} = \text{notPresent};$$

2) если клиентом в составе сообщения ClientHelloMessage был использован идентификатор предварительно распределенного ключа iPSK или ePSK, или сформированное клиентом значение VerifyMessage.sign.present равно notPresent, то клиент формирует код целостности под последовательностью октетов H_4, см. 5.5, и помещает код целостности в сообщение VerifyMessage, т. е.:

$$\text{VerifyMessage.mac.present} = \text{isPresent};$$

$$\text{VerifyMessage.mac.length} = 16;$$

$$\text{VerifyMessage.mac.code} = \text{Streebog}_{512}(H_4)[0, \ldots, 15].$$

В противном случае сервер определяет значение

$$\text{VerifyMessage.mac.present} = \text{notPresent};$$

н) после формирования сообщения VerifyMessage клиент передает его транспортному протоколу для отправки серверу в зашифрованном виде; для шифрования и контроля целостности сообщения VerifyMessage используются криптографические механизмы, указанные в поле ServerHelloMessage. algorithm, и выработанная ранее ключевая информация CHTS;

о) клиент формирует ключевую информацию для протокола передачи прикладных данных — для этого клиент формирует определенную в 5.5 последовательность октетов H_5 и применяет определенный в 5.6.2 алгоритм формирования ключевой информации.

После формирования ключевой информации клиент успешно завершает протокол выработки ключей.

5.7.4 Завершающие действия сервера

На четвертом этапе сервер завершает выполнение протокола выработки общих ключей. После получения ответа от клиента сервер выполняет следующие действия:

а) в случае получения сообщения AllertMessage сервер завершает сеанс защищенного взаимодействия. В противном случае сервер переходит к анализу полученных сообщений;

б) сервер вырабатывает ключевую информацию CHTS — для этого сервер формирует определенную в 5.5 последовательность октетов H_3 и применяет определенный в 5.6.1 алгоритм формирования ключевой информации;

в) используя алгоритм, указанный в поле ServerHelloMessage.algorithm и выработанную ключевую информацию CHTS, сервер расшифровывает, при наличии, отправленные клиентом расширения, а также полученное от клиента сообщение VerifyMessage; если при расшифровании сервер определяет нарушение целостности хотя бы одного из полученных сообщений и/или расширений, сервер отправляет клиенту сообщение AlertMessage со значением ошибки wrongIntegrityCode и завершает сеанс защищенного взаимодействия;

г) если сервером ранее направлялось расширение RequestCertificate Extension с запросом сертификата или SetCertificateExtension с указанием конкретного сертификата ключа проверки электронной подписи, или сервером было получено от клиента корректное расширение CertificateExtension, содержащее сертификат ключа проверки электронной подписи, или корректное расширение InformCertificateExtension, содержащее номер сертификата ключа электронной подписи, то сервер:

1) выбирает соответствующий сертификату ключ проверки электронной подписи Q_c;

2) проверяет, что поле VerifyMessage.sign.present принимает значение isPresent; если это не так, сервер отправляет клиенту сообщение AlertMessage с сообщением lostIntegrityCode и завершает сеанс защищенного взаимодействия;

3) с использованием ключа проверки электронной подписи Q_c проверяет истинность подписи под последовательностью октетов H_4, см. 5.5, содержащейся в VerifyMessage.sign.code, т. е. выполнение равенства

$$\text{Verify}(Q_c, \text{Streebog}_{512}(H_4), \text{VerifyMessage.sign.code}) = \text{True};$$

если равенство не выполнено, то сервер отправляет клиенту сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия;

д) если клиентом в составе сообщения ClientHelloMessage был использован идентификатор предварительно распределенного ключа iPSK или ePSK, или значение поля VerifyMessage.sign.present равно notPresent, то сервер:

1) проверяет, что поле VerifyMessage.mac.present принимает значение isPresent. Если это не так, сервер отправляет клиенту сообщение AlertMessage со значением lostIntegrityCode и завершает сеанс защищенного взаимодействия;

2) формирует код целостности под последовательностью октетов H_4, см. 5.5, и сравнивает ее значение с полученным в составе сообщения VerifyMessage, т. е. выполнимость равенства

$$\text{VerifyMessage.mac.code} = \text{Streebog}_{512}(H_4)[0, \ldots, 15];$$

если равенство не выполнено, то сервер отправляет клиенту сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия;

е) сервер формирует ключевую информацию для протокола передачи прикладных данных — для этого сервер формирует определенную в 5.5 последовательность октетов H_5 и применяет определенный в 5.6.2 алгоритм формирования ключевой информации.

После формирования ключевой информации сервер успешно завершает протокол выработки ключей.

5.8 Формирование и проверка корректности расширений

Правила формирования расширений, связанных с идентификацией и аутентификацией абонентов, изложены в 5.3 и 5.4. В этом подразделе излагаются механизмы проверки корректности данных

расширений с указанием перечня возникающих ошибок, а также принципы формирования расширений, не связанных с идентификацией и аутентификацией.

5.8.1 Расширение RequestCertificateExtension

Расширение RequestCertificateExtension формируется абонентами с целью запроса сертификата ключа проверки электронной подписи. Расширение может направляться клиентом после сообщения ClientHelloMessage и сервером после сообщения ServerHelloMessage.

После получения расширения RequestCertificateExtension абонент должен проверить его корректность, выполнив следующие проверки:

а) если поле RequestCertificateExtension.certproctype содержит значение any, а абонент не обладает сертификатом ключа проверки электронной подписи, то абонент принимает решение о некорректности сертификата и отправляет второму абоненту сообщение AlertMessage со значением wrongCertificateProcessed;

б) если поле RequestCertificateExtension.certproctype содержит значение number:

1) если поле RequestCertificateExtension.identifier содержит последовательность октетов, которая не может быть интерпретирована абонентом в качестве номера конкретного сертификата, то абонент принимает решение о некорректности сертификата и отправляет второму абоненту сообщение AlertMessage со значением wrongCertificateNumber;

2) если поле RequestCertificateExtension.identifier содержит номер сертификата, которым абонент не владеет, то абонент принимает решение о некорректности сертификата и отправляет второму абоненту сообщение AlertMessage со значением unsupportedCertificateNumber;

3) если поле RequestCertificateExtension.identifier содержит номер сертификата, которым владеет абонент, то абонент, используя описанный в 5.8.4 алгоритм, проверяет действительность указанного сертификата; в случае нарушения действительности сертификата абонент принимает решение о некорректности сертификата и отправляет сообщение AlertMessage со значением notValidCertificateNumber.

Примечание — Абонентам рекомендуется регулярно проводить проверки подлинности имеющихся у них сертификатов проверки электронной подписи. Удаление недействительных сертификатов до начала выполнения протокола выработки ключей позволит существенно сократить время его выполнения;

в) если поле RequestCertificateExtension.certproctype содержит значение issuer:

1) если поле RequestCertificateExtension.identifier содержит последовательность октетов, которая не может быть интерпретирована абонентом в качестве удостоверяющего центра, имеющего право выдавать сертификаты ключей проверки электронной подписи, то абонент принимает решение о некорректности сертификата и отправляет сообщение AlertMessage со значением wrongCertificateIssuer;

2) если абонент не владеет сертификатом ключа проверки электронной подписи, выданным удостоверяющим центром, указанным в поле SetCertificateExtension.identifier, то абонент принимает решение о некорректности сертификата и отправляет сообщение AlertMessage со значением unsupportedCertificateIssuer.

В случае, если в ходе выполненных проверок абонент принял решение о корректности полученного расширения, он выбирает сертификат ключа проверки электронной подписи для дальнейшего использования.

5.8.2 Расширение SetCertificateExtension

Расширение SetCertificateExtension формируется абонентами с целью явного указания сертификата ключа проверки электронной подписи, предполагаемого к использованию для аутентификации второго абонента. Расширение может направляться клиентом после сообщения ClientHelloMessage, сервером после сообщения ServerHelloMessage и не может отправляться одновременно с расширением RequestCertificateExtension. Данное расширение рекомендуется использовать в случае предварительного распределения сертификатов ключей проверки электронной подписи.

После получения расширения SetCertificateExtension абонент должен проверить его корректность, выполнив следующие проверки:

а) если поле SetCertificateExtension.certproctype содержит значение any, то абонент принимает решение о некорректности сертификата и отправляет второму абоненту сообщение AlertMessage со значением wrongCertificateProcessed;

б) если поле SetCertificateExtension.certproctype содержит значение number:

1) если поле SetCertificateExtension.identifier содержит последовательность октетов, которая не может быть интерпретирована абонентом в качестве номера конкретного сертификата, то абонент принимает решение о некорректности расширения и отправляет второму абоненту сообщение AlertMessage со значением wrongCertificateNumber;

2) если поле SetCertificateExtension.identifier содержит номер сертификата, которым абонент не владеет, то абонент принимает решение о некорректности расширения и отправляет второму абоненту сообщение AlertMessage со значением unsupportedCertificateNumber;

3) если поле SetCertificateExtension.identifier содержит номер сертификата, которым владеет абонент, то абонент, используя описанный в 5.8.4 алгоритм, проверяет действительность указанного сертификата; в случае нарушения действительности сертификата, абонент принимает решение о некорректности расширения и отправляет сообщение AlertMessage со значением notValidCertificateNumber.

в) если поле SetCertificateExtension.certproctype содержит значение issuer:

1) если поле RequestCertificateExtension.identifier содержит последовательность октетов, которая не может быть интерпретирована абонентом в качестве удостоверяющего центра, имеющего право выдавать сертификаты ключей проверки электронной подписи, то абонент принимает решение о некорректности сертификата и отправляет сообщение AlertMessage со значением wrongCertificateIssuer;

2) если абонент не владеет сертификатом ключа проверки электронной подписи, выданным удостоверяющим центром, указанным в поле SetCertificateExtension.identifier, то абонент принимает решение о некорректности сертификата и отправляет сообщение AlertMessage со значением unsupportedCertificateIssuer.

В случае, если в ходе выполненных проверок абонент принял решение о корректности полученного расширения, он выбирает сертификат ключа проверки электронной подписи для дальнейшего использования.

5.8.3 Расширение InformCertificateExtension

Расширение InformCertificateExtension формируется абонентами с целью указания номера сертификата ключа проверки электронной подписи, используемого абонентом для подтверждения собственной аутентичности.

Данное расширение должно использоваться в случае предварительного распределения сертификатов ключей проверки электронной подписи. Данное расширение может направляться абонентами в ходе второго и третьего этапов протокола выработки общих ключей и не может отправляться одновременно с расширением CertificateExtension.

После получения расширения InformCertificateExtension абонент должен проверить его корректность, выполнив следующие проверки:

а) если последовательность октетов InformCertificateExtension.identfier, которая не может быть интерпретирована абонентом в качестве номера конкретного сертификата, то абонент принимает решение о некорректности расширения и отправляет второму абоненту сообщение AlertMessage со значением wrongCertificateNumber;

б) если последовательность октетов InformCertificateExtension.identifier содержит номер сертификата, которым абонент не владеет, то абонент принимает решение о некорректности расширения и отправляет второму абоненту сообщение AlertMessage со значением unsupportedCertificateNumber;

в) если последовательность октетов InformCertificateExtension.identifier содержит номер сертификата, которым владеет абонент, то абонент, используя алгоритм, описанный в 5.8.4, перечисление б), проверяет действительность указанного сертификата; в случае нарушения действительности сертификата, абонент принимает решение о некорректности расширения и отправляет сообщение AlertMessage со значением notValidCertificateNumber.

В случае, если в ходе выполненных проверок абонент принял решение о корректности полученного расширения, он выбирает сертификат ключа проверки электронной подписи для дальнейшего использования.

5.8.4 Расширение CertificateExtension

Расширение CertificateExtension формируется абонентами с целью передачи другому абоненту сертификата ключа проверки электронной подписи. Данное расширение может направляться абонентами в ходе второго и третьего этапов протокола выработки общих ключей и не может отправляться одновременно с расширением InformCertificateExtension.

После получения расширения CertificateExtension абонент должен проверить его корректность, выполнив следующие проверки:

а) проверить, что поле CertificateExtension.format содержит значение, определяемое типом CertificateFormat; если поле не содержит значение из данного типа, либо указанный в данном поле формат не поддерживается абонентом, то он должен принять решение о некорректности расширения и оправить сообщение AlertMessage со значением unsupportedCertificateFormat;

б) выполнить проверку действительности сертификата, содержащегося в поле CertificateExtension. certificate; для этого необходимо выполнить следующие проверки:

1) определить удостоверяющий центр, выдавший содержащийся в поле CertificateExtension. certificate сертификат, и проверить наличие действительного сертификата удостоверяющего центра, выдавшего этот сертификат; в случае, если такого сертификата не обнаружено, либо сертификат не является действительным, то абонент должен принять решение о некорректности расширения и оправить сообщение AlertMessage со значением unsupportedCertificateIssuer.

Примечание — В случае, если цепочка сертификатов, удостоверяющих действительность полученного в расширении CertificateExtension сертификата, состоит более чем из одного сертификата, то проверка действительности должна быть проведена для каждого сертификата в цепочке. Рекомендуется проводить данную проверку до начала выполнения сеанса защищенного взаимодействия;

2) проверить, что текущее время попадает во временной интервал действия сертификата, содержащегося в поле CertificateExtension.certificate; в противном случае необходимо принять решение о некорректности расширения и оправить сообщение AlertMessage со значением expiredCertificate;

3) проверить, что область использования сертификата определена и заключается в выработке электронной подписи или в выработке ключа; если указанное утверждение неверно, то необходимо принять решение о некорректности расширения и отправить сообщение AlertMessage со значением wrongCetificateApplication;

4) с использованием сертификата удостоверяющего центра, выдавшего сертификат, содержащийся в расширении CertificateExtension, проверить истинность электронной подписи под содержащимся в расширении сертификатом; если подпись не верна, то необходимо принять решение о некорректности расширения и оправить сообщение AlertMessage со значением wrongCertificateIntegrityCode.

В случае, если в ходе выполненных проверок абонент принял решение о корректности полученного расширения, он выбирает сертификат ключа проверки электронной подписи для дальнейшего использования.

5.8.5 Расширение RequestIdentifierExtension

Расширение RequestIdentifierExtension формируется абонентами с целью указания своего идентификатора и запроса идентификатора другого абонента. Данное расширение может направляться в ходе первого и второго этапов выполнения протокола выработки общих ключей. Расширение может использоваться в случае предварительно распределенных ключей аутентификации.

Проверка корректности содержащейся в расширении последовательности октетов при получении расширения не проводится, поскольку корректность идентификатора проверяется позднее путем проверки кодов целостности, содержащихся в сообщении VerifyMessage.

5.8.6 Расширение KeyMechanismExtension

Расширение KeyMechanismExtension представляет собой средство для комплексного контроля за используемыми криптографическими механизмами. С помощью данного расширения можно установить следующие параметры транспортного протокола:

а) криптографические алгоритмы, используемые для выработки ключевой информации и производных ключей шифрования и контроля целостности передаваемой информации;

б) объем информации, шифруемой на одном ключе;

в) граничные значения для счетчиков, используемых для выработки уникальных номеров фрейма;

г) механизм формирования уникальных номеров фреймов.

Детальное описание параметров и криптографических механизмов, которые могут быть изменены с помощью расширения KeyMechanismExtension, приводится в 7.3 и приложении Г.

Расширение KeyMechanismExtension может формироваться клиентом и направляться серверу в ходе первого этапа выполнения протокола выработки ключей.

Если сервер, получивший расширение KeyMechanismExtension, не может использовать криптографические механизмы и/или параметры, определяемые значением типа KeyMechanismType, сервер должен отправить сообщение об ошибке AlertMessage со значением кода ошибки, равным unsupportedKeyMechanism, и завершить сеанс защищенного взаимодействия.

5.9 Вспомогательный алгоритм проверки точки эллиптической кривой

В состав сообщений ClientHelloMessage и ServerHelloMessage входит структура EllipticCurvePoint, содержащая в себе координаты выработанной абонентом точки эллиптической кривой.

Для проверки того, что данная точка определена корректно, получатель указанного сообщения должен выполнить следующие проверки:

а) проверить, что поле EllipticCurvePoint.id содержит идентификатор эллиптической кривой, определяемый перечислением EllipticCurveID, см. В.3.9; если поле EllipticCurvePoint.id содержит значение, отличное от определяемого перечислением EllipticCurveID, или абонент не поддерживает вычисления на эллиптической кривой с данным идентификатором, то абонент должен направить сообщение об ошибке AllertMessage со значением unsupportedEllipticCurveID;

б) если идентификатор EllipticCurvePoint.id определяет эллиптическую кривую, заданную в канонической форме Вейерштрасса, то абонент должен:

1) определить величину x значением поля EllipticCurvePoint.x;

2) определить величину y значением поля EllipticCurvePoint.y;

3) проверить выполнение сравнения $y^2 = x^3 + ax + b \pmod p$, где параметры a, b, p определены идентификатором эллиптической кривой согласно В.3.9; если указанное сравнение не выполняется, то абонент должен направить сообщение об ошибке AllertMessage со значением wrongEllipticCurvePoint;

в) если идентификатор EllipticCurvePoint.id определяет эллиптическую кривую, заданную в форме скрученной кривой Эдвардса, то абонент должен:

1) определить величину u значением поля EllipticCurvePoint.x;

2) определить величину v значением поля EllipticCurvePoint.y;

3) проверить выполнение сравнения $eu^2 + v^2 = 1 + du^2v^2 \pmod p$, где параметры e, d, p определены идентификатором эллиптической кривой согласно В.3.9; если указанное сравнение не выполняется, то абонент должен направить сообщение об ошибке AllertMessage со значением wrongEllipticCurvePoint;

г) в случае успешного выполнения одного из указанных сравнений в зависимости от значения идентификатора EllipticCurvePoint.id, определить точку P эллиптической кривой либо парой (x, y), либо (u, v);

д) в случае, если определяемый идентификатором эллиптической кривой E кофактор c, см. 5.2.2, удовлетворяет неравенству $c > 1$, то необходимо проверить, что точка cP отлична от нейтрального элемента группы точек эллиптической кривой (бесконечно удаленной точки O).

Эллиптическая кривая, определяемая значением идентификатора EllipticCurvePoint.id, и определенная в перечислении г) точка P должны быть использованы при выработке общей ключевой информации, см. 5.2.2.

6 Протокол передачи прикладных данных

6.1 Основные задачи

Основные задачи протокола передачи прикладных данных заключаются в следующем:

формировании из поступающей с прикладного уровня информации фреймов заданной длины (сообщений ApplicationDataMessage),

выработке ключевой информации, используемой для зашифрования и контроля целостности сформированных фреймов,

передаче сформированных фреймов данных и выработанной ключевой информации протоколу транспортного уровня.

Длина формируемых фреймов ограничивается параметром транспортного протокола maxFrameLength, см. 7.2, — максимально допустимой длиной фреймов, в которые вкладываются поступающие с прикладного уровня сообщения.

6.2 Ключевая система

Ключевая система протокола передачи прикладных данных состоит из следующего множества:

ключевой информации SATS и CATS, выработанной в ходе выполнения протокола выработки общих ключей; данная ключевая информация используется для выработки производных ключей шифрования и выработки имитовставки; в зависимости от объема переданных или принятых данных, ключевая информация CATS и SATS преобразуется в ходе выполнения протокола передачи прикладных данных; ее текущее состояние обозначается символами

$$SATS_n, CATS_n,$$

где n — натуральное число, принимающее значения $n = 1, 2, \ldots$; алгоритм преобразования ключевой информации описан в 6.4;

пары производных ключей сервера — ключа шифрования eSFK и ключа выработки имитовставки iSFK, предназначенных для обеспечения конфиденциальности и целостности информации, передаваемой от сервера клиенту; ключи eSFK и iSFK вырабатываются из текущего состояния ключевой информации $SATS_n$ и обозначаются символами

$$eSFK_{n, m}, iSFK_{n, m},$$

где m — натуральное число, принимающее значения m = 1, 2, ...; алгоритм выработки производных ключей сервера описан в 6.5;

пары производных ключей клиента — ключа шифрования eCFK и ключа выработки имитовставки iCFK, предназначенных для обеспечения конфиденциальности и целостности информации, передаваемой от клиента серверу; ключи eCFK и iCFK вырабатываются из текущего состояния ключевой информации $CATS_n$ и обозначаются символами

$$eCFK_{n, m}, iCFK_{n, m};$$

ключа аутентификации iPSK, который может быть выработан клиентом и сервером в ходе выполнения протокола передачи прикладных данных; данный ключ может быть использован в ходе последующего сеанса защищенного взаимодействия для аутентификации абонентов; протокол выработки ключа аутентификации iPSK описан в 6.6.

6.3 Параметры протокола

Параметрами протокола передачи прикладных данных являются следующие значения:

maxApplicationSecretCount — максимально возможное количество преобразований ключевой информации $CATS_n$ и $SATS_n$, допустимое в рамках одного сеанса защищенного взаимодействия; данная величина является максимально возможным значением счетчика числа преобразований n;

maxFrameKeysCount — максимально допустимое количество преобразований производных ключей шифрования $eSFK_{n, m}$, $eCFK_{n, m}$ и производных ключей выработки имитовставки $iSFK_{n, m}$, $iCFK_{n, m}$ для одного фиксированного состояния ключевой информации $CATS_n$ и $SATS_n$; данная величина является максимально возможным значением счетчика выработанных производных ключей m.

Примечание — Указанные параметры не могут выбираться произвольно. Значения параметров должны зависеть от используемых криптографических механизмов, см. В.3.8, класса СКЗИ, реализующего настоящие рекомендации по стандартизации, а также параметров транспортного протокола, см. 7.2. Рекомендуемые значения указанных выше величин для различных криптографических механизмов приведены в приложении Г.

6.4 Алгоритм преобразования ключевой информации

В результате выполнения протокола выработки ключей клиентом и сервером вырабатывается ключевая информация CATS и SATS, предназначенная для выработки ключей шифрования и ключей выработки имитовставки. В настоящем подразделе описывается механизм преобразования данной информации в ходе выполнения транспортного протокола.

Пусть n счетчик, принимающий значения в интервале от 1 до maxApplicationSecretCount. Счетчик n может представляться как переменная типа LengthOctet — в случае, если значение параметра maxApplicationSecretCount удовлетворяет неравенству

$$maxApplicationSecretCount \le 255,$$

либо как переменная типа LengthShortInt — в случае, если значение параметра maxApplicationSecretCount удовлетворяет неравенствам

$$256 \le maxApplicationSecretCount \le 65535.$$

Точный способ представления счетчика n определяется константой типа KeyMechanismType.

Последовательность ключевых значений $CATS_n$ и $SATS_n$ определяется равенствами

$$CATS_1 = CATS,$$

$$SATS_1 = SATS,$$

$$CATS_n = HMAC_{512}(T, CATS_{n-1}||Ser(n)),$$

$$SATS_n = HMAC_{512}(T, SATS_{n-1}||Ser(n)),$$

где CATS, SATS — ключевая информация, выработанная в ходе выполнения протокола выработки ключей, а величина T представляет собой общую для клиента и сервера ключевую информацию, определенную при

выполнении протокола выработки ключей, см. 5.6.2, а n = 2, ... , maxApplicationSecretCount. При этом длина последовательности октетов Ser(n) должна совпадать с длиной последовательности октетов, представляющей значение n, и может принимать значение 1 либо 2.

Примечания

1 Указанная выше последовательность действий схематично изображена на рисунке 6.

2 Указанный способ преобразования ключевой информации, по сути, представляет собой описанный в [2] алгоритм HKDF-Expand на основе отечественной функции хэширования Streebog, независимо примененный к ключевой информации CATS и SATS. Отличия от [2] заключаются в способе определения начального значения и использовании состоящего из двух октетов счетчика ключей.

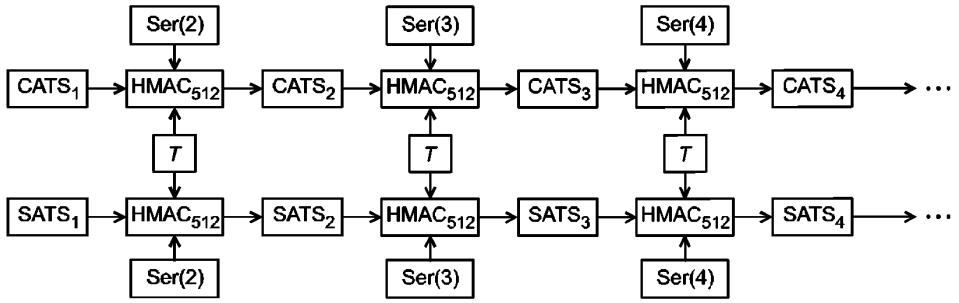

Рисунок 6 — Схема преобразования ключевой информации

При исчерпании всех допустимых значений счетчика n абонент должен направить сообщение AlertMessage с кодом ошибки keyResourceTimeUp и завершить сеанс защищенного взаимодействия. После этого, если абонент является клиентом, он может начать новое защищенное взаимодействие, инициировав выполнение протокола выработки ключей. Если абонент является сервером, то он должен перейти в состояние, ожидающее начало выполнения протокола выработки ключей.

6.5 Алгоритмы выработки производных ключей

Производные ключи вырабатываются из ключевой информации $CATS_n$ и $SATS_n$ для каждого значения счетчика и используются непосредственно для шифрования и контроля целостности передаваемых по каналам связи фреймов.

Пусть m счетчик, принимающий значения в интервале от 0 до maxFrameKeysCount. Счетчик m может представляться как переменная типа LengthOctet — в случае, если значение параметра maxFrameKeysCount удовлетворяет неравенству

$$\text{maxFrameKeysCount} \leq 255,$$

либо как переменная типа LengthShortInt — в случае, если значение параметра maxFrameKeysCount удовлетворяет неравенствам

$$256 \leq \text{maxFrameKeysCount} \leq 65535.$$

Точный способ представления счетчика m определяется константой типа KeyMechanismType.

Алгоритм выработки производных ключей шифрования $eCFK_{n,m}$, $eSFK_{n,m}$ зависит от алгоритма блочного шифрования, определяемого значением поля ServerHelloMessage.algorithm, см. 5.7.2, и определяется для каждого фиксированного значения n следующими равенствами

$$eCFK_{n,0} = CATS_n\,[0, \ldots, 31],$$

$$eCFK_{n,m} = ACPKM(eCFK_{n,m-1}),$$

$$eSFK_{n,0} = SATS_n\,[0, \ldots, 31],$$

$$eSFK_{n,m} = ACPKM(eSFK_{n,m-1}),$$

где m = 1, ... , maxFrameKeysCount, а отображение ACPKM, предназначенное для преобразования ключевой информации, определяется в Р 1323565.1.017—2018 следующим образом:

а) для алгоритма блочного шифрования «Магма» значение параметра J определяется равенством J = 4; для алгоритма «Кузнечик» — равенством J = 2;

б) отображение ACPKM определяется равенством

$$K_m = ACPKM(K_{m-1}) = E(K_{m-1}, D_1)||\cdots||E(K_{m-1}, D_J),$$

где K_m — ключ, вырабатываемый из ключа K_{m-1}, а $D_1||\cdots||D_J$ — константная последовательность октетов, определяемая следующим образом:

D = (0x80||0x81||0x82||0x83||0x84||0x85||0x86||0x87||
0x88||0x89||0x8A||0x8B||0x8C||0x8D||0x8E||0x8F||
0x90||0x91||0x92||0x93||0x94||0x95||0x96||0x97||
0x98||0x99||0x9A||0x9B||0x9C||0x9D||0x9E||0x9F),

и

D[0] = 0x80;
D[1] = 0x81;
...
D[30] = 0x9E;
D[31] = 0x9F;

Алгоритм выработки ключей выработки имитовставки $iCFK_{n,m}$, $iSFK_{n,m}$ соответственно, клиента и сервера также зависит от алгоритма блочного шифрования, определяемого значением поля ServerHelloMessage.algorithm, см. 5.7.2, и определяется следующим образом.

Для алгоритма блочного шифрования «Магма» значение натурального числа Ctr определяется равенством

$$Ctr = 18446744069414584320_{10} = FFFFFFFF00000000_{16},$$

а для алгоритма «Кузнечик» — равенством

$$Ctr = 340282366920938463444927863358058659840_{10} = FFFFFFFFFFFFFFFF0000000000000000_{16}.$$

Тогда

$$CK_n = CATS_n\ [32, \ldots, 63],$$

$$SK_n = SATS_n\ [32, \ldots, 63],$$

$$iCFK_{n,m} = E(CK_n, Ser(Ctr + mJ))||\cdots||E(CK_n, Ser(Ctr + (m+1)J - 1)),$$

$$iSFK_{n,m} = E(SK_n, Ser(Ctr + mJ))||\cdots||E(CK_n, Ser(Ctr + (m+1)J - 1)),$$

где величина J определена выше в перечислении а).

Примечание — Описанные выше алгоритмы выработки ключей клиента — производных ключей шифрования $eCFK_{n,m}$ и ключей выработки имитовставки $iCFK_{n,m}$ схематично изображены на рисунке 7. Схема выработки ключей сервера — производных ключей шифрования $eSFK_{n,m}$ и ключей выработки имитовставки $iSFK_{n,m}$ аналогична изображенной на рисунке 7.

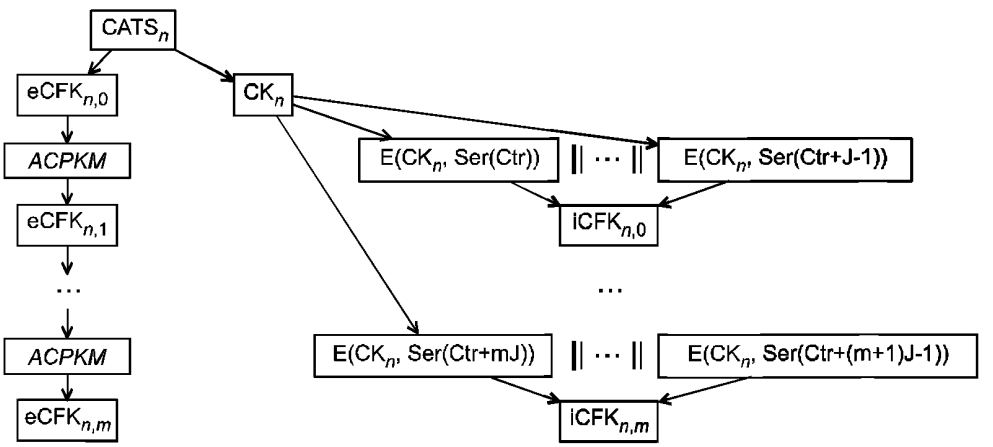

Рисунок 7 — Схема выработки производных ключей

6.6 Протокол выработки ключа аутентификации iPSK

Выработка ключа аутентификации iPSK представляет собой отдельный протокол, выполняемый в рамках протокола передачи прикладных данных.

Целью выработки ключа аутентификации iPSK является снижение объема информации, передаваемой в ходе выполнения следующего сеанса протокола выработки ключей, с сохранением свойства аутентификации абонентов. Выполнение протокола выработки ключа аутентификации не является обязательным.

Протокол выработки ключа аутентификации iPSK инициируется клиентом путем формирования сообщения GeneratePSKMessage и отправки данного сообщения серверу.

Для формирования сообщения GeneratePSKMessage клиент выполняет следующие действия:

а) вырабатывает случайную последовательность октетов $Rand_c$ длиной 32 октета и помещает ее в поле GeneratePSKMessage.random;

б) помещает в поле GeneratePSKMessage.id.present значение notPresent.

Сформированное сообщение передается клиентом транспортному протоколу для отправки в канал связи в зашифрованном виде. При этом для зашифрования сообщения GeneratePSKMessage и контроля его целостности используются текущие производные ключи $eCFK_{n,m}$, $iCFK_{n,m}$ и текущие значения счетчиков n, m.

После получения сообщения GeneratePSKMessage сервер выполняет следующие действия:

в) вырабатывает случайную последовательность октетов $Rand_s$ длиной 32 октета и помещает ее в поле GeneratePSKMessage.random;

г) вырабатывает последовательность октетов PSK, определяемую равенством

$$PSK = HMAC_{512}(T, Rand_s||Rand_c||ID_s||ID_c^*||FN_s||FN_c),$$

где T — общая для клиента и сервера ключевая информация, используемая для преобразования ключевой информации, см. 6.4;

$Rand_c$, $Rand_S$ — случайные октеты, выработанные ранее, соответственно, клиентом и сервером;

ID_s, ID_c — идентификаторы сервера и клиента (опционально), использованные клиентом и сервером в процессе выполнения протокола выработки ключей;

FN_s, FN_c — уникальные номера фреймов, в которых клиент отправил свой запрос на инициализацию протокола выработки ключа аутентификации iPSK и сервер планирует отправить свой ответ на запрос клиента;

д) определяет идентификатор выработанного ключа iPSK равенством

$$ID_{iPSK} = PSK[0, \ldots, 31];$$

е) определяет ключ аутентификации iPSK равенством

$$iPSK = PSK[32, \ldots, 63];$$

ж) формирует поле GeneratePSKMessage.id равенствами:

$$GeneratePSKMessage.id.present = isPresent,$$

$$GeneratePSKMessage.id.length = 32,$$

$$GeneratePSKMessage.id.id = ID_{iPSK}.$$

Сформированное сообщение передается сервером транспортному протоколу для отправки в канал связи в зашифрованном виде. При этом для зашифрования сообщения GeneratePSKMessage и контроля его целостности используют текущие производные ключи $eCFK_{n,m}$, $iCFK_{n,m}$ и значения счетчиков n, m, использованные при формировании уникального номера FN_s.

После получения ответа от сервера клиент также вычисляет значение ключа аутентификации iPSK. Для этого он выполняет следующие действия:

з) вырабатывает последовательность октетов PSK, определяемую равенством

$$PSK = HMAC_{512}(T, Rand_s||Rand_c||ID_s||ID_c^*||FN_s||FN_c),$$

где T — общая для клиента и сервера ключевая информация, используемая для преобразования ключевой информации, см. 6.4;

$Rand_c$, $Rand_s$ — случайные октеты, выработанные ранее, соответственно клиентом и сервером;

ID_s, ID_c — идентификаторы сервера и клиента (опционально), использованные клиентом и сервером в процессе выполнения протокола выработки ключей;

FN_s, FN_c — уникальные номера фреймов, в которых клиент и сервер отправляли сообщения GeneratePSKMessage;

и) определяет идентификатор выработанного ключа iPSK равенством

$$ID_{iPSK} = PSK[0, \ldots, 31]$$

и сравнивает полученное значение со значением, содержащимся в поле GeneratePSKMessage. id.id; если данные значения не совпадают, то клиент направляет серверу в зашифрованном виде сообщение AlertMessage со значением ошибки, равным wrongInternalPSKIdentifier;

к) если значения идентификаторов ключа совпали, то клиент определяет ключ аутентификации iPSK равенством

$$iPSK = PSK[32, \ldots, 63]$$

и завершает выполнение протокола выработки ключа аутентификации iPSK.

7 Протокол транспортного уровня

7.1 Основные сведения

Протокол транспортного уровня предназначен для обеспечения конфиденциальности и целостности фреймов информации, передаваемых в рамках защищенного взаимодействия между абонентами.

Сообщения, сформированные протоколами сеансового уровня — протоколом выработки ключей и протоколом передачи прикладных данных, передаются транспортному протоколу для отправки в канал связи. В ходе выполнения транспортного протокола сообщения вкладываются во фреймы, см. В.4.1, которые, при необходимости, шифруются, дополняются кодом целостности и направляются в канал связи.

Выработка ключевой информации, используемой для шифрования и контроля целостности передаваемых сообщений, выполняется протоколами сеансового уровня. Информация о ключах и используемых криптографических механизмах передается протоколу транспортного уровня протоколом сеансового уровня. После получения фреймов из канала связи проводится проверка их целостности, расшифрование и передача содержащихся в фреймах сообщений протоколу сеансового уровня. Канал связи, с которым взаимодействует протокол транспортного уровня, может быть реализован с помощью различных протоколов сетевого уровня. Настоящие рекомендации не накладывают ограничений на использование протоколов сетевого уровня.

Примечание — Сетевой протокол, используемый для реализации канала связи, не обязательно должен соответствовать сетевому уровню взаимодействия модели ВОС. Для реализации канала связи допускается использование как протоколов, находящихся выше сетевого уровня в модели ВОС, например TCP/IP, так и протоколов находящихся ниже.

В случае, если используемый для организации канала связи сетевой протокол не гарантирует поточную передачу данных (это подразумевает, что данные, первыми отправленные одним абонентом с сеансового уровня, будут первыми получены на сеансовом уровне другим абонентом), реализация протокола транспортного уровня должна обеспечивать данную возможность.

Для реализации контроля за последовательностью передаваемых фреймов настоящими рекомендациями вводится механизм нумерации передаваемых в канал связи фреймов. Данный механизм позволяет не только присвоить фрейму его уникальный номер, см. В.3.13, но и связать фрейм с криптографическими ключами, обеспечивающими конфиденциальность и целостность содержащихся во фрейме сообщений.

Настоящие рекомендации используют параметрический подход к выработке уникальных номеров фреймов. Значения используемых параметров существенным образом зависят от:

а) используемого протокола сетевого уровня взаимодействия;

б) криптографических требований по безопасности, устанавливающих максимальный объем зашифровываемой на одном ключе информации.

Детальное рассмотрение вопросов формирования уникальных номеров фреймов и механизмов связи уникальных номеров фреймов с криптографическими ключами содержится в 7.2 и 7.3.

Настоящими рекомендациями не вводится обязательное требование гарантированной доставки фреймов, передаваемых в ходе выполнения протокола передачи прикладных данных. Если данное свойство является необходимым, оно также должно обеспечиваться реализацией протокола транспортного уровня.

При реализации методов обеспечения свойств поточной передачи данных и гарантированной доставки сообщений должны применяться не криптографические механизмы, при этом повторное шифрование содержащейся в фреймах информации для различных значений синхропосылок и ключевой информации не допускается. Описание методов обеспечения указанных выше свойств выходит за рамки настоящих рекомендаций и должно регламентироваться соответствующими нормативными документами отдельно для каждого типового механизма реализации канала связи.

7.2 Параметры протокола

Параметрами протокола транспортного уровня, а также алгоритма формирования последовательности октетов FrameNumber, содержащей уникальный номер фрейма, являются следующие значения:

maxFrameLength — максимально допустимая длина последовательности октетов, являющейся сериализованным представлением структуры данных Frame (максимальная длина фрейма);

maxFrameCount — максимально допустимое количество фреймов, конфиденциальность и целостность которых обеспечивается одной парой ключей шифрования и выработки имитовставки.

Примечание — Указанные параметры не могут выбираться произвольно. С целью удовлетворения криптографическим требованиям по безопасности, должно выполняться условие

$$maxFrameLength * maxFrameCount \le B_{max},$$

где B_{max} — максимально возможный размер последовательности октетов, которая может быть зашифрована на одном ключе. Данное значение зависит от используемых криптографических механизмов, см. В.3.8, класса СКЗИ, реализующего настоящие рекомендации по стандартизации, и не может превосходить величины, определенной в Р 1323565.1.005—2017. Рекомендуемые значения указанных величин для различных криптографических механизмов приведены в приложении Г.

7.3 Алгоритм формирования уникальных номеров фреймов

Уникальный номер фрейма, представляющий собой последовательность из пяти октетов, см. В.3.13, позволяет однозначно связать фрейм с криптографическими ключами, используемыми для обеспечения конфиденциальности и целостности передаваемых данных.

Для связывания фрейма с криптографическими ключами используется набор из трех целых неотрицательных чисел (счетчиков):

а) l — число, принимающее значения в интервале от 0 до maxFrameCount-1, определяет количество фреймов, зашифрованных на одной паре ключей шифрования и выработки имитовставки. Значение счетчика l для первого фрейма полагается равным 0 и увеличивается на единицу для каждого последующего переданного фрейма. При изменении значений счетчиков m и n значение счетчика l обнуляется.

При передаче незашифрованных сообщений начальное значение счетчика l также полагается равным 0 для сообщений ClientHelloMessage и ServerHelloMessage, см. 5.7.1 и 5.7.2, и увеличивается на единицу для каждого последующего незашифрованного фрейма;

б) m — число, принимающее значения в интервале от 0 до maxFrameKeysCount, где величина maxFrameKeysCount определяет количество допустимых преобразований производных ключей шифрования и выработки имитовставки. При изменении значения счетчика n значение счетчика m обнуляется;

в) n — число, принимающее значения в интервале от 0 до maxApplicationSecretCount, где величина maxApplicationSecretCount определяет количество допустимых преобразований ключевой информации, производимых в ходе выполнения протокола передачи прикладных данных.

Примечания

1 Значения величин m, n передаются протоколу транспортного уровня протоколами сеансового уровня и не изменяются в ходе формирования и проверки фреймов.

2 При выполнении протокола выработки ключей значение величины m равно нулю для сообщений, передаваемых в открытом виде, и равно единице для сообщений, передаваемых в зашифрованном виде.

3 При выполнении протокола выработки ключей значение величины n всегда равно нулю. При выполнении протокола передачи прикладных данных значение величины n начинает изменяться со значения n = 1.

4 Значения счетчиков изменяются независимо для клиента и сервера.

Максимальные значения для констант, ограничивающих сверху величины указанных счетчиков, приводятся в приложении Г. Размер памяти, выделяемой под хранение значений указанных счетчиков, определяется значением константы типа KeyMechanismType, определяющей текущий набор параметров криптографических механизмов.

Уникальный номер фрейма (последовательность октетов FrameNumber) формируется из значений указанных счетчиков следующим образом:

а) если текущий используемый криптографический механизм определяет, что счетчики l, m есть переменные типа LengthShortInt, а счетчик n есть переменная типа LengthOctet, то значение уникального номера number определяется равенствами:

number[0] = n,
number[1] = m[0],
number[2] = m[1],
number[3] = l[0],
number[4] = l[1];

б) если текущий используемый криптографический механизм определяет, что счетчик l есть переменная типа LengthOctet, а счетчики m, n есть переменные типа LengthShortInt, то значение уникального номера number определяется равенствами:

number[0] = n[0];
number[1] = n[1];
number[2] = m[0];
number[3] = m[1];
number[4] = l.

7.4 Алгоритм формирования фрейма

Для формирования фрейма, см. В.4.1, необходимы следующие параметры, передаваемые протоколом сеансового уровня:

а) сформированное на сеансовом уровне сообщение или расширение message, представленное в виде последовательности октетов (сериализованного представления одной из определенных в В.5 и В.6 структур данных);

б) тип сообщения или расширения, см. определитель MessageType;

в) длина сообщения или расширения Len(message) — целое неотрицательное число, не превосходящее $2^{16} - 1$;

г) идентификатор криптографического механизма, используемого для контроля целостности и, при необходимости, зашифрования помещаемого во фрейм сообщения или расширения, см. определитель CryptoMechanism;

д) два целых неотрицательных числа, определяющих значения счетчиков n и m;

е) пара ключей — ключ выработки имитовставки $iK \in B^{32}$ и, при необходимости, ключ шифрования $eK \in B^{32}$;

ж) не обязательная дополнительная информация adata, представленная в виде последовательности октетов длины Len(adata), не превосходящей 64. Данная информация, при ее наличии, будет помещена во фрейм в открытом виде, а ее целостность будет обеспечена используемым криптографическим механизмом.

Примечание — При формировании фрейма необязательная дополнительная информация adata помещается в заголовок фрейма и может служить, например для идентификации канала связи.

7.4.1 Процедура вложения

После проверки перечисленных выше параметров на корректность выполняется последовательность действий:

а) формируется заголовок — часть фрейма, всегда передаваемая в незашифрованном виде:

1) поле tag принимает значение

$$ftype + 4*hlen,$$

где ftype полагается равным plainFrame, для фреймов, передаваемых в незашифрованном виде, или encryptedFrame, для фреймов, передаваемых в зашифрованном виде, а hlen определяется равенством

$$hlen = 8 + Len(adata)$$

и обозначает длину заголовка.

Необходимость того, должен ли быть зашифрован фрейм, определяется идентификатором криптографического механизма, см. В.3.8, и значениями счетчиков n, m, см. 7.3.

Примечание — Согласно 5.7.1 и 5.7.2 в незашифрованном виде могут передаваться сообщения, формируемые в ходе выполнения первого и второго этапов протокола выработки ключей, а также сообщения об ошибках;

2) согласно 7.4.3 формируется дополнение padding — последовательность октетов длины Len(padding);

3) поле length полагается равным

$$hlen + 3 + Len(message) + Len(padding) + Len(icode),$$

где icode — сериализованное представление кода целостности, см. В.3.12. Длина кода целостности однозначно определяется значением используемого идентификатора криптографического механизма.

Примечание — Величина hlen+3 представляет собой сумму 8 обязательных октетов заголовка, длину необязательной информации, помещаемой в заголовок, а также одного октета, отводимого под хранение типа сообщения message, и двух октетов, отводимых под сохранение длины сообщения message;

4) поле number полагается равным уникальному номеру фрейма;

5) в поле adata помещается передаваемая в открытом виде не обязательная дополнительная информация;

б) формируется тело фрейма, содержащее сообщение message и дополнение. Данная часть фрейма, как правило, передается в зашифрованном виде:

1) поле type полагается равным типу помещаемого во фрейм сообщения;

2) поле meslen полагается равным значению Len(message);

3) в поле message помещается собственно передаваемое сообщение;

4) в поле padding помещается дополнение;

в) если сообщение должно передаваться в открытом виде, то, с использованием функции хеширования или функции вычисления имитовставки и ключа iK, формируется код целостности или имитовставка под частью фрейма, начинающейся с поля type и оканчивающейся окончанием дополнения, которая помещается в поле icode. Алгоритм выработки кода целостности или имитовставки определяется идентификатором криптографического механизма, см. В.3.8;

г) если сообщение должно передаваться в зашифрованном виде, то применяется процедура зашифрования и вычисления имитовставки, описываемая в следующем разделе.

Сформированный фрейм передается в канал связи.

7.4.2 Процедура зашифрования и вычисления имитовставки

Процедура зашифрования сообщения и вычисления имитовставки формируемого фрейма зависит от идентификатора используемого криптографического механизма.

В случае, если идентификатор используемого криптографического механизма равен magmaCTRplusOMAC или kuznechikCTRplusOMAC, то:

а) для сформированного фрейма, начиная с его начала и заканчивая окончанием дополнения, вычисляется значение имитовставки с использованием режима выработки имитовставки, регламентируемого ГОСТ Р 34.13—2015, раздел 5.6, блочного шифра, определяемого идентификатором криптографического механизма, и ключа выработки имитовставки iK; данное значение помещается в поле icode;

б) после вычисления имитовставки фрейм зашифровывается, начиная с поля type и заканчивая окончанием дополнения, с использованием режима гаммирования, регламентируемого ГОСТ Р 34.13—2015, раздел 5.2, блочного шифра, определяемого идентификатором криптографического механизма, и ключа шифрования eK;

в) для блочного шифра «Кузнечик» (значение идентификатора — kuznechikCTRplusOMAC) в качестве синхропосылки должна выступать последовательность октетов, сформированная первыми восьмью октетами заголовка (начиная с нулевого и заканчивая седьмым октетом);

г) для блочного шифра «Магма» (значение идентификатора — magmaCTRplusOMAC) в качестве синхропосылки должна выступать последовательность октетов, сформированная четырьмя октетами заголовка (начиная с четвертого и заканчивая седьмым октетом).

Примечание — В случае использования блочного шифра «Магма» согласно ГОСТ Р 34.13—2015, раздел 5.2, синхропосылка должна состоять из четырех октетов, что меньше, чем длина уникального номера фрейма. Этот факт приводит к необходимости ограничения общего числа фреймов, отправляемых и получаемых в ходе выполнения одного сеанса защищенного взаимодействия, величиной 2^{32}. Данное ограничение учтено при выборе рекомендуемых значений параметров защищенного взаимодействия, см. приложение Г.

В случае если идентификатор используемого криптографического механизма равен magmaAEAD или kuznechikAEAD, то:

а) вычисление имитовставки и зашифрование сообщений выполняются с помощью режима, осуществляющего одновременно шифрование и аутентификацию (режим AEAD), см. Р 1323565.1.026—2019, блочного шифра, определяемого идентификатором криптографического механизма, ключа шифрования eK и ключа выработки имитовставки iK;

б) заголовок фрейма является «ассоциированными данными», а тело фрейма — данными, которые подлежат зашифрованию; вычисленное в ходе выполнения режима, осуществляющего одновременное шифрование и аутентификацию, значение имитовставки помещается в поле icode;

Способ выработки синхропосылки зависит от алгоритма блочного шифрования, определяемого идентификатором криптографического механизма:

в) в случае блочного шифра «Магма» (значение идентификатора — magmaAEAD) в качестве синхропосылки должен выступать двоичный вектор длины 63 бита, являющийся результатом однократного сдвига в сторону младших разрядов вектора, образованного первыми восемью октетами заголовка (начиная с нулевого и заканчивая седьмым октетом); т. е.

$$iv = (frame[0]||...||frame[7]) >> 1;$$

где frame это последовательность октетов, являющаяся сериализованным представлением сформированного фрейма, см. В.4.1;

г) в случае блочного шифра «Кузнечик» (значение идентификатора — kuznechikAEAD) в качестве синхропосылки должен выступать двоичный вектор длины 127 бит, в котором первые восемь октетов совпадают с первыми восемью октетами заголовка (начиная с нулевого и заканчивая седьмым октетом), а оставшиеся октеты принимают нулевые значения, т. е.:

```
iv[0] = frame[0];
.....
iv[7] = frame[7];
iv[8] = 0;
.....
iv[15] = 0;
```

Схематично, механизм зашифрования помещаемого во фрейм сообщения и вычисления кода целостности фрейма изображен на рисунке 8.

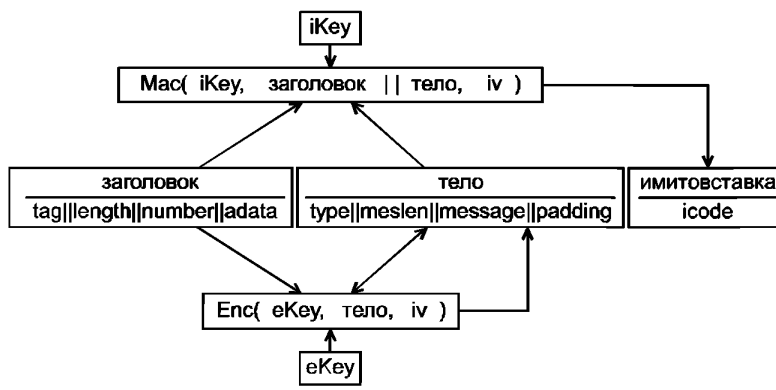

Рисунок 8 — Схема зашифрования и вычисления кода целостности фрейма

Примечание — На рисунке 8 символами eKey и iKey обозначены ключи, соответственно, шифрования и выработки имитовставки, а символом iv — синхропосылка, вырабатываемая в соответствии с рекомендациями, изложенными в перечислениях в) и г).

7.4.3 Процедура генерации дополнения

Дополнение представляет собой последовательность октетов длины Len(padding). Последовательность вкладывается во фрейм одновременно с сообщением. Основная причина использования дополнения заключается в сокрытии длины передаваемого сообщения.

Для выработки дополнения необходимо использовать генератор случайных (псведо-случайных) чисел. Используемый генератор случайных чисел не должен допускать возможности восстановления прослушивающим канал нарушителем значения дополнения, например с использованием случайных данных, передаваемых в открытом виде в составе сообщений ClientHelloMessage и/или ServerHelloMessage, см. 5.7.1 и 5.7.2. Использование в качестве дополнения последовательности фиксированных значений не рекомендуется.

При выборе длины дополнения Len(padding) рекомендуется применять одну или несколько стратегий, зависящих от типа передаваемого сообщения и потребностей приложений прикладного уровня:

а) длина дополнения может выбираться равной нулю, т. е. дополнение может отсутствовать; такую стратегию выбора дополнения рекомендуется применять для сообщений, передаваемых в открытом виде, а также в ситуациях, когда сокрытие длины передаваемых сообщений полагается излишним;

б) длина дополнения может выбираться таким образом, чтобы длина фрейма была постоянной (константой) для всех фреймов, независимо от типа вкладываемого во фрейм сообщения; при этом значение константы может выбираться так, чтобы максимизировать пропускную способность канала связи;

в) длина дополнения может выбираться случайным образом так, чтобы длина фрейма была кратной длине блока используемого алгоритма блочного шифрования; это может оказаться существенным при эффективной реализации алгоритмов шифрования и вычисления имитовставки;

г) длина дополнения может выбираться минимально возможной величиной, дополняющей длину фрейма до величины, кратной длине блока используемого алгоритма блочного шифрования; при такой стратегии размер дополнения минимизируется;

д) длина дополнения может выбираться случайно.

Во всех перечисленных случаях длина дополнения Len(padding) должна удовлетворять условию

$$hlen + 3 + Len(message) + Len(padding) + Len(icode) < maxFrameLength,$$

которое, согласно 7.4.1, следует из определения суммарной длины фрейма.

7.5 Алгоритм расшифрования фрейма

При получении фреймов из канала связи протокол транспортного уровня должен проверить их целостность, при необходимости расшифровать и передать полученные сообщения протоколу прикладного уровня.

Последовательность действий, которые выполняет абонент при получении фрейма, зависит от того, зашифрован ли данный фрейм, а также от значения уникального номера фрейма.

Для определения криптографических механизмов, используемых для контроля целостности и расшифрования полученных фреймов, должен использоваться идентификатор криптографического механизма. Значение данного идентификатора может быть определено при получении сообщений ClientHelloMessage и ServerHelloMessage, см. 5.7.1 и 5.7.2, либо установлено или изменено протоколом сеансового уровня.

7.5.1 Действия при получении незашифрованного фрейма

Согласно описанию протокола выработки общих ключей, см. 5.7, первый фрейм, который получает абонент, должен содержать незашифрованное сообщение. Вслед за ним могут передаваться фреймы, содержащие как незашифрованные, так и зашифрованные сообщения или расширения.

При извлечении из фрейма незашифрованного сообщения или расширения, абонент должен выполнить следующие действия:

а) определить, что полученный фрейм передан в незашифрованном виде. Для этого абонент должен проверить выполнение равенства

frame[0] (mod 4) = plainFrame.

Также, согласно 7.3, должно быть проверено выполнение равенств:

frame[3] = 0;

. . .

frame[6] = 0;

frame[7] = number,

где number это уникальное для каждого незашифрованного фрейма натуральное число;

б) определить длину заголовка hlen = frame[0] >> 2;

в) если frame[hlen-1] = 0, то определить значение идентификатора криптографического механизма, см. В.3.8, используя для этого октеты frame[hlen+3] и frame[hlen+4]. В случае если frame[hlen-1] > 0, то использовать идентификатор криптографического алгоритма, определенный ранее;

г) согласно значению идентификатора криптографического механизма определить длину кода целостности L. Проверить, что для полученного фрейма данное значение удовлетворяет условиям:

$$frame[idx - 1] = isPresent,$$

$$frame[idx] = L,$$

где idx = Len(frame) – L, а Len(frame) — длина всего фрейма, определяемая значениями октетов frame[1] и frame[2], см. В.4.1;

д) согласно значению идентификатора криптографического механизма вычислить значение кода целостности или имитовставки от фрагмента фрейма, начинающегося с frame[0] и заканчивающегося frame[idx − 2], где значение idx определено в перечислении г).

Примечания

1 В случае вычисления имитовставки идентификатор используемого предварительно распределенного ключа должен быть получен путем разбора вложенного во фрейм сообщения ClientHelloMessage или ServerHelloMessage, см. 5.7.1 и 5.7.2.

2 Для сообщения AlertMessage идентификатор алгоритма выработки кода целостности должен определяться значением поля algorithm;

е) проверить, что полученное значение кода целостности или имитовставки совпадает со значением, содержащимся во фрагменте полученного фрейма, начинающемся с frame[idx + 1] и заканчивающемся окончанием фрейма, где значение idx определено в перечислении г).

В случае если одна из перечисленных выше проверок не выполняется, то абонент должен отправить незашифрованное сообщение об ошибке AlertMessage со значением wrongIntegrityCode и завершить выполнение сеанса защищенного взаимодействия.

Если все перечисленные проверки успешно пройдены, то протокол транспортного уровня должен передать протоколу сеансового уровня начинающееся с frame[hlen+3] сообщение или расширение message, длина которого определяется значением полей frame[hlen+1] и frame[hlen+2], а также тип сообщения или расширения, определяемый значением поля frame[hlen].

7.5.2 Действия при получении зашифрованных фреймов

При получении зашифрованного фрейма должна быть выполнена последовательность действий:

а) определить, что полученный фрейм передан в зашифрованном виде. Для этого абонент должен проверить выполнение равенства

$$frame[0] \pmod 4 = encryptedFrame.$$

б) определить длину заголовка hlen = frame[0] >> 2;

в) согласно значению идентификатора криптографического механизма определить длину кода целостности L. Проверить, что для полученного фрейма данное значение удовлетворяет условиям:

$$frame[idx - 1] = isPresent,$$

$$frame[idx] = L,$$

где idx = Len(frame) − L, а Len(frame) — длина всего фрейма, определяемая значениями октетов frame [1] и frame [2], см. В.4.1;

г) используя значения полей frame [3], ... , frame [7] определить значения счетчиков n, m, см. 7.3, и пару ключей шифрования и выработки имитовставки, однозначно связанных с вычисленными значениями счетчиков;

д) используя значение идентификатора криптографического механизма, ключи шифрования и выработки имитовставки, а также процедуры, описанные в 7.4.2, расшифровать фрагмент полученного фрейма, начинающийся с frame[hlen+3] и заканчивающийся frame[idx − 2], где значение idx определено в перечислении в);

е) вычислить значение имитовставки от фрагмента фрейма, начинающегося с frame [0] и заканчивающегося frame[idx−2], где значение idx определено в перечислении в).

Примечание — В случае режима шифрования с одновременной выработкой имитовставки, см. В.3.8, перечисления г) и д) должны выполняться одновременно;

ж) проверить, что полученное значение имитовставки совпадает со значением, содержащимся во фрагменте полученного фрейма, начинающемся с frame[idx+1] и заканчивающемся окончанием фрейма, где значение idx определено в перечислении в).

В случае если одна из перечисленных выше проверок не выполняется, то абонент должен отправить зашифрованное сообщение об ошибке AlertMessage со значением wrongIntegrityCode и завершить выполнение сеанса защищенного взаимодействия.

Если все перечисленные проверки успешно пройдены, то протокол транспортного уровня должен передать протоколу сеансового уровня начинающееся с frame[hlen+3] сообщение или расширение message, длина которого определяется значением полей frame[hlen+1] и frame[hlen+2], а также тип сообщения или расширения, определяемый значением поля frame[hlen].

Приложение А
(справочное)

Типовые схемы реализации протокола выработки ключей с аутентификацией абонентов

А.1 Схема аутентификации на основе предварительно распределенного ключа

Схема с аутентификацией на основе предварительно распределенного ключа может применяться в устройствах, использующих предварительно распределенный секретный ключ PSK для взаимной аутентификации клиента и сервера. Механизмы выработки предварительно распределенных ключей для данного класса устройств приведены в приложении Б.

С точки зрения сеансового уровня схема протокола выработки ключей с аутентификацией на основе предварительно распределенного ключа изображена на рисунке А.1.

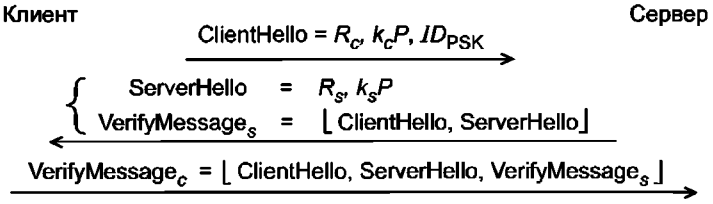

Рисунок А.1 — Схема аутентификации на основе предварительно распределенного ключа. Сеансовый уровень

В данной схеме символом ID_{PSK} обозначается идентификатор предварительно распределенного секретного ключа PSK, известного абонентам до момента начала выполнения протокола выработки ключей. В качестве ключа PSK могут выступать ключи ePSK или iPSK, определенные в 5.2.1.

Символами R_c, R_s обозначаются случайные последовательности октетов длины 32 октета, вырабатываемые, соответственно клиентом и сервером при формировании сообщений ClientHelloMessage и ServerHelloMessage. Символами k_cP, k_sP обозначаются случайные точки некоторой фиксированной эллиптической кривой, вырабатываемые, соответственно клиентом и сервером при формировании сообщений ClientHelloMessage и ServerHelloMessage.

На сеансовом уровне сообщения рассматриваются как сериализованные представления вводимых спецификацией защищенного взаимодействия структур данных. С точки зрения сеансового уровня сообщения передаются в ходе выполнения протокола выработки ключей в незашифрованном виде и без имитовставок, подтверждающих целостность передаваемых сообщений.

Зашифрование передаваемых сообщений и вычисление для них имитовставки производится на транспортном уровне защищенного взаимодействия. Это приводит к схеме обмена сообщениями, полностью отражающей выполняемые процедуры зашифрования и контроля целостности передаваемых сообщений и изображенной на рисунке А.2.

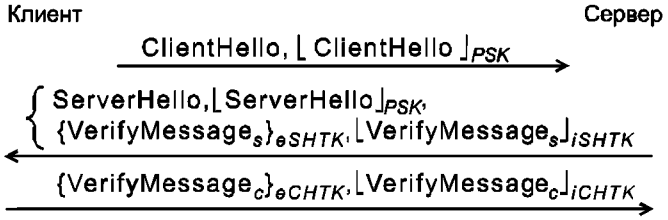

Рисунок А.2 — Схема аутентификации на основе предварительно распределенного ключа. Транспортный уровень

В данной схеме символами eSHTS и eCHTS обозначаются ключи шифрования, а символами iSHTS и iCHTS обозначаются ключи имитозащиты, используемые для обеспечения конфиденциальности и целостности информации, передаваемой от сервера к клиенту и, соответственно от клиента к серверу.

Примечание — Формально протоколы, использующие рассматриваемую схему протокола выработки ключей, различаются в зависимости от того, какой из предварительно распределенных ключей ePSK или iPSK используется. Различие состоит в способе формирования константы R_2, используемой в алгоритме выработки ключевой информации SHTS и CHTS, см. 5.5.

А.2 Схема аутентификации на основе ключа проверки электронной подписи

Схема с аутентификацией на основе ключа проверки электронной подписи предназначена для контрольных и измерительных устройств, срок службы которых превышает срок действия предварительно распределенных ключей аутентификации. Для аутентификации используются ключи электронной подписи и ключи проверки электронной подписи, сертификаты которых не известны абонентам до начала выполнения протокола.

Рассматриваемая схема в основном применима для класса устройств, целью которых является предоставление услуги доступа к защищенному взаимодействию различным физическим лицам — обладателям пары асимметричных ключей аутентификации.

При этом в качестве ключей аутентификации выступают ключ электронной подписи и ключ проверки электронной подписи.

С точки зрения сеансового уровня схема протокола выработки ключей с аутентификацией на основе ключа проверки электронной подписи изображена на рисунке А.3.

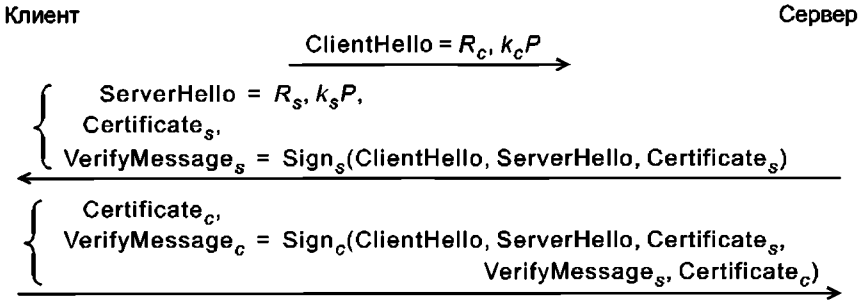

Рисунок А.3 — Схема аутентификации на основе ключа проверки электронной подписи. Сеансовый уровень

Как и ранее, приведенная схема не учитывает факт передачи части сообщений в зашифрованном виде, а также не содержит в себе передаваемых имитовставок. Поскольку зашифрование и вычисление имитовставки производится на транспортном уровне защищенного взаимодействия, полная схема взаимодействия, отражающая реально выполняемые процедуры зашифрования и контроля целостности передаваемых сообщений, изображена на рисунке А.4.

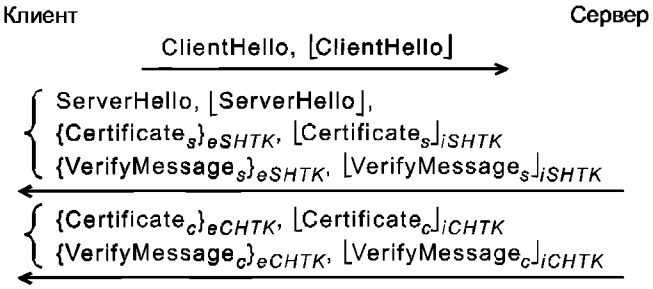

Рисунок А.4 — Схема аутентификации на основе ключа проверки электронной подписи. Транспортный уровень

А.3 Схема аутентификации на основе предварительно распределенных ключей проверки электронной подписи

Схема со взаимной аутентификацией на основе предварительно распределенных ключей проверки электронной подписи может применяться в устройствах, использующих для взаимной аутентификации сервера и клиента ключи проверки электронной подписи, которые не передаются абонентами в ходе выполнения протокола, а распределены заранее и известны абонентам до начала выполнения протокола.

С точки зрения сеансового уровня данная схема протокола выработки ключей изображена на рисунке А.5.

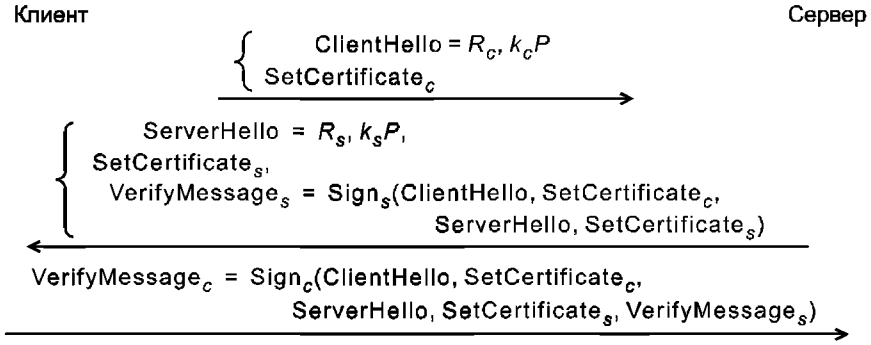

Рисунок А.5 — Схема со взаимной аутентификацией на основе предварительно распределенных ключей проверки электронной подписи. Сеансовый уровень

Как и ранее, приведенная схема не учитывает факт передачи части сообщений в зашифрованном виде, а также не содержит в себе передаваемых имитовставок. Поскольку зашифрование и вычисление имитовставки производится на транспортном уровне защищенного взаимодействия, полная схема взаимодействия, отражающая реально выполняемые процедуры зашифрования и контроля целостности передаваемых сообщений, изображена на рисунке А.6 следующим образом.

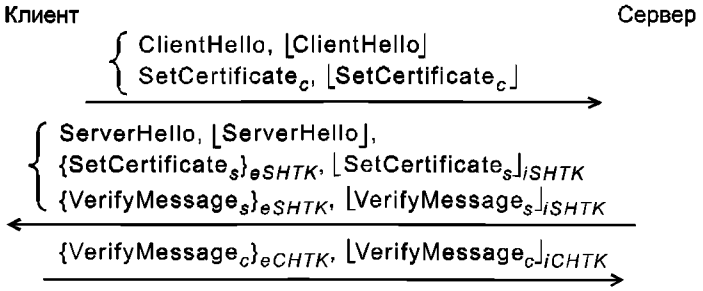

Рисунок А.6 — Схема со взаимной аутентификацией на основе предварительно распределенных ключей проверки электронной подписи. Транспортный уровень

Приложение Б
(справочное)

Механизмы формирования предварительно распределенных ключей

Б.1 Основные положения

Спецификацией криптографических механизмов защищенного взаимодействия контрольных и измерительных устройств порядок выработки предварительно распределенных ключей аутентификации ePSK и соответствующих им идентификаторов ID_{ePSK} не регламентируется.

Настоящее приложение содержит один из допустимых механизмов формирования предварительно распределенных ключей аутентификации, являющийся модификацией подхода, регламентируемого Р 1323565.1.019—2018.

Предполагается, что при взаимодействии контрольные и измерительные устройства могут выступать в роли клиентов и передавать информацию единому центру, в качестве которого выступает сервер, а также могут взаимодействовать между собой по типу «каждый с каждым», выступая как в роли клиента, так и в роли сервера.

Б.2 Идентификация участников защищенного взаимодействия

Для идентификации участников защищенного взаимодействия контрольных и измерительных устройств используется двухуровневая схема, расширяющая множество идентификаторов, вводимых в 5.3. Каждый участник защищенного взаимодействия обладает следующими идентификаторами:

а) идентификатором первого уровня, в качестве которого должен выступать идентификатор клиента ID_c либо идентификатор сервера ID_s, определяемый спецификацией криптографических механизмов защищенного взаимодействия; наличие идентификатора первого уровня, согласно 5.3, является обязательным.

Примечание — Интерпретация значения, содержащегося в идентификаторе первого уровня, должна зависеть от конкретной ситуации применения контрольных и измерительных устройств. Например, при реализации защищенного взаимодействия серии контрольных и измерительных устройств, передающих информацию единому центру (серверу), идентификатор первого уровня может содержать информацию о производителе устройств, номере серии устройств, дате ввода серии устройств в эксплуатацию и т.п. С другой стороны, при реализации связи типа «каждый с каждым» идентификатор первого уровня может являться уникальным номером устройства;

б) идентификатором второго уровня, в качестве которого может выступать последовательность октетов произвольной конечной длины, обозначаемая символом ID_{cp} для клиента и символом — ID_{sp} для сервера; идентификатор второго уровня является опциональным, т. е. он может быть не определен, и, в случае определения, должен принимать уникальные значения в рамках одного контрольного или измерительного устройства.

Примечания

1 Интерпретация значения, содержащегося в идентификаторе второго уровня также должна зависеть от конкретной ситуации применения контрольных и измерительных устройств. В случае реализации защищенного взаимодействия серии контрольных и измерительных устройств, передающих информацию единому центру (серверу), идентификатор второго уровня может содержать номер устройства, дату выпуска, максимальный срок эксплуатации и т. п. Данный идентификатор может иметь ограниченный временной интервал и изменяться в ходе эксплуатации устройства, например при замене содержащегося в устройстве блока СКЗИ.

С другой стороны, при реализации связи типа «каждый с каждым» идентификатор второго уровня может определять уникальный номер процесса или пользователя устройства.

2 Поскольку идентификаторы второго уровня являются опциональными, их применение целесообразно в рамках системы защищенного взаимодействия, поддерживающей единый формат (размер и содержание) и механизмы распознавания идентификаторов второго уровня; пример такой системы рассматривается в Р 1323565.1.019—2018.

Б.3 Операции в конечном поле $F2^{256}$

Каждый двоичный вектор из V^{256} может быть представлен в виде элемента конечного поля $F2^{256}$. Данное представление взаимно однозначно и может быть задано следующим образом.

Пусть вектор $a = (a_0, \ldots, a_{255})$ из V^{256}, тогда ему будет соответствовать многочлен $a(x) = \sum_{i=0}^{255} a_i x^i \in F_2[x]$, $\deg(a(x)) < 256$. Используя данное соответствие определим операции сложения и умножения двоичных векторов из V^{256} следующим образом.

Определим неприводимый многочлен

$$p(x) = x^{256} + x^{10} + x^5 + x^2 + 1 \in V_2[x].$$

Рассмотрим произвольные $a = (a_0, \ldots, a_{255})$, $b = (b_0, \ldots, b_{255}) \in V^{256}$, а также соответствующие им многочлены

$$a(x)=\sum_{i=0}^{255} a_i x^i, \quad b(x)=\sum_{i=0}^{255} b_i x^i, \quad a(x), b(x) \in F_2[x],$$

тогда

вектор a + b определяется равенством a + b = c где c = (c_0, ..., c_{255}) есть вектор, которому соответствует многочлен

$$c(x) = a(x) + b(x) \pmod{p(x)},$$

где mod p(x) обозначает взятие остатка от деления многочлена a(x) + b(x) на многочлен p(x);

вектор a × b определяется равенством a × b = c, где c = (c_0, ..., c_{255}) есть вектор, которому соответствует многочлен

$$c(x) = a(x) \times b(x) \pmod{p(x)},$$

где mod p(x) обозначает взятие остатка от деления многочлена a(x) × b(x) на многочлен p(x).

Б.4 Ключевая система

Ключевая система, используемая при взаимодействии контрольных и измерительных устройств, представляет собой вариант схемы распределения ключей Блома, см. [3], и состоит из следующих секретных ключей, используемых для выработки уникальных ключей аутентификации.

Б.4.1 Мастер-ключ

Пусть u натуральное число. Мастер-ключ МК представляет собой симметричную матрицу A={a_{ij}} размера (u + 1) × (u + 1), где

$$a_{ij} = a_{ji}, \ 0 \le i \le u, \ 0 \le j \le u, \text{ и } a_{ij} \in V^{256}.$$

Значение параметра u зависит от максимального срока действия ключевой системы, используемой для выработки ключей аутентификации. По умолчанию, значение u полагается равным u = 2048.

Примечания

1 Максимальное количество различных элементов матрицы A равно

$$1/2 \times (u + 1) \times (u + 2).$$

2 Коэффициенты матрицы A однозначно связаны с многочленом:

$$f(x,y)=\sum_{i=0}^{u}\sum_{j=0}^{u} a_{ij} x^i y^j \in F_{2^{256}}[x,y].$$

3 Значение мастер-ключа является криптографически опасной информацией и не должно быть известно участникам защищенного взаимодействия. Формирование мастер-ключа и вырабатываемой из него ключевой информации должно производиться уполномоченной на данную деятельность организацией. Если СКЗИ планируется использовать в областях, регулируемых в соответствии с действующим законодательством и нормативными актами, то формирование мастер-ключа должно производиться в соответствии с Р 1323565.1.012—2017.

Б.4.2 Ключи, соответствующие идентификаторам первого уровня

Идентификаторам первого уровня, см. Б.2, соответствуют следующие ключи:

а) идентификатору клиента ID_c соответствует уникальный ключ клиента K_c, зависящий от мастер-ключа МК и идентификатора сервера ID_c. Ключ K_c представляет собой вектор:

$$(b_0, \ldots, b_u), \ b_i \in V^{256}, \ i = 0, \ldots, u,$$

определяемый равенством

$$f\left(x, Streebog_{256}(ID_c)\right) = \sum_{i=0}^{u} b_i x^i \in F_{2^{256}}[x],$$

где многочлен f(x, y) определен ранее в Б.4.1.

Эквивалентным определением величин $b_i \in V^{256}$ является равенство

$$b_i = \sum_{j=0}^{u} a_{ij} (Streebog_{256}(ID_c))^j,$$

выполненное для всех i = 0, ... , u.

Примечание — Значение ключа клиента K_c должно быть известно только клиенту и может использоваться им для формирования ключей аутентификации непосредственно в ходе выполнения протокола выработки ключей. Формирование ключа клиента K_c должно производиться непосредственно после выработки мастер-ключа МК в условиях, указанных в Б.4.1;

б) идентификатору сервера ID_s соответствует уникальный ключ сервера K_s, зависящий от мастер-ключа МК и идентификатора сервера ID_s. Ключ K_s представляет собой вектор:

$$(c_0, \ldots, c_u),\ c_j \in V^{256},\ j = 0, \ldots, u,$$

определяемый равенством

$$f(Streebog_{256}(ID_s), y) = \sum_{i=0}^{u} c_j y^j \in F_{2^{256}}[x],$$

где многочлен f(x, y) определен ранее в Б.4.1.

Эквивалентным определением величин $c_j \in V^{256}$ является равенство

$$c_j = \sum_{i=0}^{u} a_{ij} \left(Streebog_{256}(ID_s)\right)^i,$$

выполненное для всех j = 0, ... , u.

Примечание — Значение ключа сервера K_s должно быть известно только серверу и может использоваться им для формирования ключей аутентификации непосредственно в ходе выполнения протокола выработки ключей. Формирование ключа сервера K_s должно производиться непосредственно после выработки мастер-ключа МК в условиях, указанных в Б.4.1.

Б.4.3 Ключ аутентификации ePSK

Ключ аутентификации ePSK представляет собой двоичную последовательность длины 256 бит и зависит от следующих значений:

мастер ключа МК, см. Б.4.1;

идентификаторов первого уровня — идентификатора клиента ID_c и идентификатора сервера ID_s, см. Б.2;

идентификаторов второго уровня — идентификатора клиента ID_{cp} и идентификатора сервера ID_{sp}, при их наличии, см. Б.2.

Для определения ключа ePSK введем промежуточный ключ K*, удовлетворяющий равенству

$$K^* = f\left(Streebog_{256}(ID_s), Streebog_{256}(ID_c)\right) = \sum_{i=0}^{u}\sum_{j=0}^{u} (Streebog_{256}(ID_s))^i \left(Streebog_{256}(ID_c)\right)^j.$$

Примечание — Значение промежуточного ключа K* может быть вычислено клиентом без знания мастер-ключа — для этого достаточно знания ключа клиента K_c, поскольку из определения многочлена f(x,y) следует равенство

$$K^* = \sum_{i=0}^{u} b_i \left(Streebog_{256}(ID_s)\right)^i.$$

Аналогично, сервер может вычислить промежуточный ключ K*, используя равенство

$$K^* = \sum_{j=0}^{u} c_j \left(Streebog_{256}(ID_c)\right)^j.$$

Ключ ePSK определяется равенством ePSK = $HMAC_{256}(K^*,R_3)$ где последовательность октетов R_3 определяется как конкатенация

$$R_3 = ID_{cp}^*||0x00||ID_{sp}^*||0x00||0x01.$$

Примечания

1 Символом «*» отмечены последовательности октетов, которые могут отсутствовать.

2 Значение ключа аутентификации ePSK должно быть известно только двум абонентам, участвующим в защищенном взаимодействии. Формирование ключа аутентификации может производиться в СКЗИ либо уполномоченной на данную деятельность организацией в соответствии с Р 1323565.1.012—2017.

В зависимости от метода хранения ключевой информации в СКЗИ идентификатор ID_{ePSK} ключа ePSK может определяться одним из следующих способов:

а) $ID_{ePSK} = ID_c||ID_{cp}||ID_s||ID_{sp}$;

б) $ID_{ePSK} = Streebog_{256}(ID_c||ID_{cp}||ID_s||ID_{sp})$.

Первый способ определения идентификатора ID_{ePSK} позволяет серверу вычислять значение ключа ePSK в процессе выполнения протокола выработки ключей. Второй способ позволяет скрыть информацию о контрольном или измерительном устройстве, передаваемую в открытом виде в составе сообщения ClientHelloMessage.

Приложение В
(обязательное)

Форматы передаваемых данных

В.1 Основные положения

В приложении приводится формальное описание форматов данных, используемых при реализации криптографических механизмов защищенного взаимодействия. Данные представляются в виде формальных описаний (типов), согласно принятой в языке Си нотации, см. [4]. Указанные структуры данных предполагаются упакованными, т. е. занимающими в памяти вычислительного средства объем, являющийся в точности суммой объемов памяти, занимаемых элементами структур данных, см. [4].

При передаче в канал связи данные сериализуются в последовательности октетов фиксированной длины, в которых нумерация начинается с младших адресов и заканчивается старшими. В канал связи передаются последовательности октетов.

В.2 Базовые типы данных

Базовые типы данных представляют собой последовательности октетов конечной длины и не требуют явного описания процесса сериализации.

В.2.1 Описатель Octet

typedef unsigned char Octet

Описатель Octet определяет минимально возможную единицу передаваемых по каналу связи данных — один октет.

В.2.2 Описатель OctetString

typedef octet *OctetString

Описатель OctetString определяет последовательность октетов произвольной конечной длины. Данный описатель используется для определения сообщений или их фрагментов без учета внутренней структуры.

OctetString рассматривается как нумерованная последовательность октетов, у которой в начале (слева) располагаются элементы с младшими номерами, а в конце (справа) — элементы со старшими номерами: такой способ представления соответствует естественному представлению массивов данных в памяти вычислительного средства.

В.2.3 Описатель LengthOctet

typedef Octet LengthOctet

Описатель LengthOctet определяет тип данных, состоящий из одного октета. Данный тип данных предназначен для определения целых неотрицательных чисел, как правило, длин последовательностей, принимающих значения от 0 до 255.

В.2.4 Описатель LengthShortInt

typedef Octet LengthShortInt[2]

Описатель LengthShortInt определяет тип данных, состоящий из последовательности октетов длины два. Данный тип данных предназначен для определения целых неотрицательных чисел, как правило, длин последовательностей, не превосходящих $65535 = 2^{16} - 1$.

Описатель LengthShortInt определяет целое неотрицательное число l, принадлежащее интервалу от 0 до $2^{16} - 1$, согласно следующему правилу

$$l = 256 \cdot len[0] + len[1],$$

где len является последовательностью типа LengthShortInt.

Процесс сериализации целого неотрицательного числа l, принадлежащего интервалу от 0 до $2^{16} - 1$, в последовательность октетов len типа LengthShortInt, заключается в выполнении равенств:

$$len[0] = l - \frac{l(mod\ 256)}{256},$$

$$len[1] = l(mod\ 256).$$

Примечание — При сериализации целых неотрицательных чисел используется сетевой порядок следования октетов.

В.2.5 Описатель RandomOctetString

typedef Octet RandomOctetString[32]

Описатель RandomOctetString определяет тип данных, состоящий из последовательности октетов фиксированной длины 32 октета. Данный тип данных используется для хранения и передачи по каналам связи данных, выработанных с использованием генераторов случайных чисел.

В.3 Перечислимые и служебные типы

В.3.1 Описатель FrameType

```
typedef enum {
  plainFrame = 0x00U,
  encryptedFrame = 0x02U
} FrameType
```

Описатель FrameType предназначен для определения типа передаваемого фрейма данных, см. В.4.1. Указанные выше константы описывают следующие возможности:

а) plainFrame — фрейм передается в незашифрованном виде;

б) encryptedFrame — фрейм передается в зашифрованном виде.

При сериализации объект типа FrameType представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

В.3.2 Описатель PresentType

```
typedef enum {
  notPresent = 0xB0,
  isPresent = 0xB1
} PresentType
```

Описатель PresentType предназначен для указания: установлено ли значение некоторой опциональной переменной или нет. Используется только в опциональных переменных. Указанные выше константы описывают следующие возможности:

а) notPresent — переменная не используется и ее значение не определено;

б) isPresent — переменная используется и ее значение определено.

При сериализации объект типа PresentType представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

В.3.3 Описатель RequestType

```
typedef enum {
  notRequested = 0xB0,
  isRequested = 0xB1
} PresentType
```

Описатель RequestType предназначен для установки запроса на получение идентификатора абонента. Указанные выше константы описывают следующие возможности:

а) notRequested — запрос идентификатора не проводится;

б) isRequested — запрос идентификатора выполняется.

При сериализации объект типа RequestType представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

В.3.4 Описатель CertificateFormat

```
typedef enum {
  plain = 0x10,
  x509 = 0x19,
  cvc = 0x20
} CertificateFormat
```

Описатель CertificateFormat предназначен для указания формата используемого сертификата ключа проверки электронной подписи, см. 5.2.1. Указанные выше константы описывают следующие возможности:

а) plain — сертификатом ключа проверки электронной подписи является последовательность октетов, полученная в результате сериализации объекта типа EllipticCurvePoint, см. В.3.10;

б) x509 — сертификат ключа проверки электронной подписи соответствует Р 1323565.1.023—2018;

в) cvc — сертификат ключа проверки электронной подписи соответствует формату сертификатов, рекомендуемых к применению в контрольных и измерительных устройствах.

При сериализации объект типа CertificateFormat представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

В.3.5 Описатель CertificateProcessedType

```
typedef enum {
  any = 0x00,
  number = 0x10,
  issuer = 0x20
} CertificateProcessedType
```

Описатель CertificateProcessedType предназначен для указания предлагаемого к использованию сертификата ключа проверки электронной подписи. Указанные выше константы описывают следующие возможности:

а) any — предполагается к использованию любой действительный сертификат ключа проверки электронной подписи;

б) number — предполагается к использованию действительный сертификат ключа проверки электронной подписи с заданным номером;

в) issuer — предполагается к использованию любой действительный сертификат ключа проверки электронной подписи с заданным центром сертификации (с явным указанием субъекта, выдавшего предполагаемый к использованию сертификат ключа проверки электронной подписи).

При сериализации объект типа CertificateProcessedType представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

В.3.6 Описатель Certificate

typedef OctetString Certificate

Описатель Certificate определяет последовательность октетов произвольной конечной длины, предназначенную для представления сертификата ключа проверки электронной подписи, см. 5.2.1. Допускается использование сертификатов ключей проверки электронной подписи, хранящихся в различных форматах. Перечень допустимых форматов определяется в В.3.4.

Примечание — Сертификат ключа проверки электронной подписи передается от одного абонента к другому при помощи расширения CertificateExtension, см. 5.7.2 и 5.7.3. Поскольку длина расширения CertificateExtension указывается при его вложении во фрейм, см. В.4.1, это позволяет однозначно восстановить длину передаваемого сертификата ключа проверки электронной подписи.

В.3.7 Описатель MessageType

```
typedef enum {
  clientHello = 0x11,
  serverHello = 0x12,
  verify = 0x13,
  applicationData = 0x14,
  alert = 0x15,
  generatePSK = 0x16,
  extensionRequestCertificate = 0x21,
  extensionCertificate = 0x22,
  extensionSetCertificate = 0x23,
  extensionInformCertificate = 0x24,
  extensionRequestIdentifer = 0x25,
  extensionKeyMechanism = 0x26
} MessageType
```

Описатель MessageType предназначен для указания типа сообщения, вкладываемого во фрейм передачи данных. Каждому типу соответствует сообщение со своей собственной формальной структурой представления данных. Указанные выше константы описывают следующие структуры данных:

а) clientHello — структура ClientHelloMessage, см. В.5.1, определяющая формат сообщения, с помощью которого клиент инициализирует выполнение защищенного взаимодействия;

б) serverHello — структура ServerHelloMessage, см. В.5.2, определяющая формат сообщения, с помощью которого сервер передает ответ на инициализирующий запрос клиента;

в) verify — структура VerifyMessage, см. В.5.3, определяющая формат сообщения, содержащего один или несколько кодов аутентификации;

г) applicationData — структура ApplicationDataMessage, см. В.5.4, определяющая формат сообщений, передаваемых в ходе выполнения протокола передачи прикладных данных;

д) alert — структура AlertMessage, см. В.5.5, определяющая формат сообщения, содержащего код ошибки выполнения протокола;

е) generatePSK — структура GeneratePSKMessage, см. В.5.6, определяющая формат сообщения, передаваемого в процессе протокола выработки ключа аутентификации;

ж) extensionRequestCertificate — структура RequestCertificateExtension, см. В.6.1, определяющая формат расширения, используемого для запроса сертификата ключа проверки электронной подписи;

з) extensionCertificate — структура CertificateExtension, см. В.6.2, определяющая формат расширения, используемого для передачи сертификатов ключей проверки электронной подписи;

и) extensionSetCertificate — структура SetCertificateExtension, см. В.6.3, определяющая формат расширения, используемого для указания использования конкретного сертификата ключа проверки электронной подписи;

к) extensionInformCertificate — структура InformCertificateExtension, см. В.6.4, определяющая формат расширения, используемого для информирования о номере используемого сертификата ключа проверки электронной подписи;

л) extensionRequestIdentifier — структура RequestIdentifierExtension, см. В.6.5, определяющая формат расширения, используемого для запроса и/или указания используемого идентификатора абонента;

м) extensionKeyMechanism — структура KeyMechanismExtension, см. В.6.6, определяющая криптографические механизмы выработки производных ключей.

При сериализации объект типа MessageType представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

В.3.8 Описатель CryptoMechanism

```
typedef enum {
  streebog256 = 0x0013,
  streebog512 = 0x0023,
  magmaGOST3413ePSK = 0x2051,
  kuznechikGOST3413ePSK = 0x2052,
  magmaGOST3413iPSK = 0x3101,
  kuznechikGOST3413iPSK = 0x3102,
  hmac256ePSK = 0x2033,
  hmac512ePSK = 0x2043,
  hmac256iPSK = 0x3033,
  hmac512iPSK = 0x3043,
  magmaCTRplusHMAC256 = 0x1131,
  magmaCTRplusGOST3413 = 0x1151,
  kuznechikCTRplusHMAC256 = 0x1132,
  kuznechikCTRplusGOST3413 = 0x1152,
  magmaAEAD = 0x1201,
  kuznechikAEAD = 0x1202,
} CryptoMechanism
```

Описатель CryptoMechanism предназначен для указания клиентом и сервером криптографических преобразований, используемых для обеспечения конфиденциальности и целостности передаваемой ими информации.

Указанные константы представляют собой целые неотрицательные числа от 0 до $2^{16} - 1$, см. В.2.4, и сформированы в соответствии со следующим правилом — значение константы представляет собой двоичный вектор длины шестнадцать, в котором фиксированные биты определяют конкретный вид криптографического алгоритма и его параметры в соответствии с рисунком В.1.

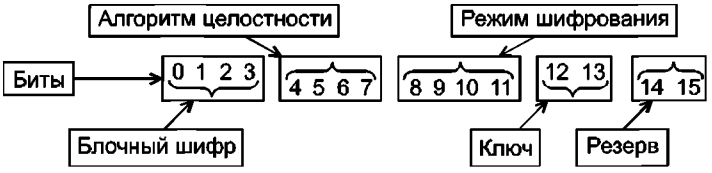

Рисунок В.1 — Порядок назначения значений бит в определении криптографического механизма

1 Для используемого алгоритма блочного шифрования допускаются следующие значения параметров:

а) 0x0 — алгоритм блочного шифрования не определен;

б) 0x1 — используется алгоритм блочного шифрования «Магма» с длиной блока 64 бита, регламентируемый ГОСТ Р 34.12—2015;

в) 0x2 — используется алгоритм блочного шифрования «Кузнечик» с длиной блока 128 бит, регламентируемый ГОСТ Р 34.12—2015;

г) 0x3 — алгоритм блочного шифрования не используется (конфиденциальность данных не обеспечивается).

2 Для используемого алгоритма вычисления кода целостности допускаются следующие значения параметров:

а) 0x0 — алгоритм вычисления кода целостности не определен;

б) 0x1 — используется бесключевая функция хеширования «Стрибог» с длиной кода 256 бит, регламентируемая ГОСТ Р 34.11—2012;

в) 0x2 — используется бесключевая функция хеширования «Стрибог» с длиной кода 512 бит, регламентируемая ГОСТ Р 34.11 — 2012;

г) 0x3 — используется ключевая функция хеширования $HMAC_{256}$ с длиной кода 256 бит, регламентируемая Р 50.1.13—2016;

д) 0x4 — используется ключевая функция хеширования $HMAC_{512}$ с длиной кода 512 бит, регламентируемая Р 50.1.13—2016;

е) 0x5 — используется ключевая функция хеширования, регламентируемая ГОСТ Р 34.13—2015.

3 Для используемого режима работы блочного шифра допускаются следующие значения параметров:

а) 0x0 — режим работы блочного шифра не определен;

б) 0x1 — режим гаммирования блочного шифра, регламентируемый ГОСТ Р 34.13 —2015;

в) 0x2 — режим работы блочных шифров, обеспечивающий одновременное шифрование и аутентификацию в соответствии с Р 1323565.1.026—2019.

4 Для параметра, определяющего используемые ключи криптографических преобразований, допускаются следующие значения параметров:

а) 0x0 — криптографические ключи не определены;

б) 0x1 — используются производные ключи шифрования, см. 6.5; данный флаг должен быть установлен для криптографических механизмов, используемых при передаче прикладных данных;

в) 0x2 — используется ключ аутентификации ePSK, см. 5.2.1;

г) 0x3 — используется ключ аутентификации iPSK, см. 5.2.1.

5 Старшие (зарезервированные) биты константы, определяющей криптографический механизм, полагаются равными нулю.

При сериализации объект типа CryptoMechanism представляется в виде последовательности октетов аналогично типу данных LengthShortInt.

В.3.9 Описатель EllipticCurveID

```
typedef enum {
  tc26_gost3410_2012_256_paramsetA = 0x01,
  tc26_gost3410_2012_256_paramsetB = 0x02,
  tc26_gost3410_2012_256_paramsetC = 0x03,
  tc26_gost3410_2012_256_paramsetD = 0x04,
  tc26_gost3410_2012_512_paramsetA = 0x05,
  tc26_gost3410_2012_512_paramsetB = 0x06,
  tc26_gost3410_2012_512_paramsetC = 0x07,
} EllipticCurveID
```

Описатель EllipticCurveID предназначен для указания параметров эллиптической кривой, используемых в протоколе выработки общих ключей, см. 5.2.2. Под параметрами эллиптической кривой подразумевается следующий набор значений:

а) простое число p;

б) коэффициенты эллиптической кривой:

либо пара a, $b \in F_p$, определяющая эллиптическую кривую E, заданную в канонической форме Вейерштрасса сравнением

$$y^2 \equiv x^3 + ax + b(\mathrm{mod}\ p),$$

либо пара e, $d \in F_p$, определяющая эллиптическую кривую E, заданную в форме скрученной кривой Эдвардса сравнением

$$eu^2 + v^2 \equiv 1 + du^2v^2(\mathrm{mod}\ p);$$

в) точка эллиптической кривой P, заданная парой своих координат;

либо пара x, $y \in F_p$ — для эллиптической кривой, заданной в канонической форме Вейерштрасса,

либо пара u, $v \in F_p$ — для эллиптической кривой, заданной в форме скрученной кривой Эдвардса;

г) простое число q, определяющее порядок подгруппы, порожденной точкой P;

д) кофактор c — натуральное число такое, что произведение cq определяет порядок всей группы точек эллиптической кривой E.

Указанные при определении типа EllipticCurveID константы определяют наборы параметров эллиптических кривых, регламентируемые следующими документами:

Р 1323565.1.024—2019, раздел 3 — эллиптические кривые, заданные в канонической форме Вейерштрасса

```
  tc26_gost3410_2012_256_paramsetB = 0x02,
  tc26_gost3410_2012_256_paramsetC = 0x03,
  tc26_gost3410_2012_256_paramsetD = 0x04,
  tc26_gost3410_2012_512_paramsetA = 0x05,
  tc26_gost3410_2012_512_paramsetB = 0x06;
```

Р 1323565.1.024—2019, раздел 4 — эллиптические кривые, заданные в форме скрученной кривой Эдвардса

- id-tc26-gost-3410-2012-256-paramSetA,
- id-tc26-gost-3410-2012-512-paramSetC;

Примечание — Поскольку указанные при определении типа EllipticCurveID константы однозначно связаны с конкретными наборами параметров эллиптической кривой, это позволяет избавиться от необходимости передавать в ходе выполнения защищенного взаимодействия информацию о размерах параметров и форме эллиптической кривой (канонической формой Вейерштрасса или формой скрученной кривой Эдвардса).

При сериализации объект типа EllipticCurveID представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

В.3.10 Описатель EllipticCurvePoint

```
typedef struct {
  EllipticCurveID id;
  OctetString x;
  OctetString y;
} EllipticCurvePoint
```

Описатель EllipticCurvePoint определяет структуру данных, предназначенную для хранения точки эллиптической кривой, заданной двумя координатами. Данная структура состоит из следующих полей:

а) id — идентификатор эллиптической кривой, которой принадлежит точка; значение идентификатора определяется типом EllipticCurveID;

б) x — x-координата точки кривой, заданной в канонической форме Вейерштрасса, либо u-координата точки кривой, заданной в форме скрученной кривой Эдвардса;

в) y — y-координата точки кривой, заданной в канонической форме Вейерштрасса, либо u-координата точки кривой, заданной в форме скрученной кривой Эдвардса.

При сериализации объект типа EllipticCurvePoint преобразуется в последовательность октетов следующим образом

point = id||x||y,

и

point[0] = id;
point[1] = x[0];
.....
point[l] = x[l-1];
point[l+1] = y[0];
.....
point[2l] = y[l-1].

где l определяется в зависимости от используемой эллиптической кривой либо равенством l = 32 (для 256-битных эллиптических кривых), либо l = 64 (для 512-битных эллиптических кривых).

При этом предполагаем, что переменные x и y типа OctetString представляют собой результат сериализации вычетов x и y, которые удовлетворяют равенствам

$$x = \sum_{i=0}^{l-1} x[i] \times 256^i, \quad y = \sum_{i=0}^{l-1} y[i] \times 256^i.$$

B.3.11 Описатель PreSharedKeyID

typedef struct {
  PresentType present;
  KeyType type;
  LengthOctet length;
  OctetString id;
} PreSharedKeyID

Описатель PreSharedKeyID определяет структуру данных, предназначенную для хранения и передачи по каналам связи идентификаторов предварительно распределенных симметричных ключей. Структура PreSharedKeyID состоит из следующих полей:

а) present — флаг того, определен ли данный идентификатор; если значение флага равно notPresent, то идентификатор не определен;

б) type — тип предварительно распределенного ключа; может принимать значения ePSKKey и iPSKKey;

в) length — длина идентификатора (количество октетов, занимаемых полем id);

г) id — произвольная последовательность октетов конечной длины, определяемой значением поля length.

Примечание — Если в качестве предварительно распределенного ключа выступает ключ iPSK (предварительно распределенный ключ, выработанный в ходе выполнения предыдущего сеанса защищенного взаимодействия), то его идентификатором является последовательность октетов фиксированной длины 32 октета, см. 6.6.

При сериализации объект типа PreSharedKeyID преобразуется в последовательность октетов следующим образом:

keyid = present,

если present = notPresent, либо

keyid = present||type||length||id,

если present = isPresent и
keyid[0] = present;
keyed[1] = type;
keyid[2] = length;
keyid[3] = id[0];
.....
keyid[length + 3] = id[length – 1]

B.3.12 Описатель IntegrityCode

typedef struct {
  PresentType present;
  LengthOctet length;
  OctetString code;
} IntegrityCode

Описатель IntegrityCode определяет структуру данных, предназначенную для передачи кодов целостности или имитовставок, вырабатываемых в ходе криптографического взаимодействия абонентов для обеспечения и проверки целостности передаваемой информации.

Структура InegrityCode состоит из следующих полей:

present — флаг того, определен ли данный код целостности или имитовставка; если значение флага равно notPresent, то значение не определено;

length — длина кода целостности (имитовставки) (количество октетов, занимаемых полем code);

code — произвольная последовательность октетов конечной длины, определяемой значением поля length. Точное значение длины кода целостности или имитовставки определяется выбранным клиентом или сервером криптографическим алгоритмом, см. тип данных CryptoMechanism.

При сериализации объект типа IntegrityCode преобразуется в последовательность октетов следующим образом

icode = present,

если present = notPresent, либо

icode = present||length||code,

если present = isPresent и

icode[0] = present;
icode[1] = length;
icode[2] = code[0];
.....
icode[length + 1] = code[length – 1]

B.3.13 Описатель FrameNumber

typedef Octet FrameNumber[5]

Описатель FrameNumber определяет последовательность из пяти октетов, предназначенную для указания криптографических номеров фреймов, см. В.4.1. Данный тип позволяет однозначно связать фрейм с криптографическими ключами, используемыми для обеспечения его конфиденциальности и целостности.

Последовательность из пяти октетов, образующих объект типа FrameNumber, может допускать различную интерпретацию, зависящую от выбранного клиентом механизма выработки производных ключей — допустимые варианты таких механизмов определяются значениями типа KeyMechanismType. Детальное описание алгоритма формирования номера фрейма содержится в 7.3.

B.3.14 Описатель KeyMechanismType

```
typedef enum {
  baseKeyMechanismMagma = 0x14,
  baseKeyMechanismKuznechik = 0x34,
  shortKCmagma = 0x18,
  shortKCkuznechik = 0x38,
  longKCmagma = 0x19,
  longKCkuznechik = 0x39,
  shortKAmagma = 0x1c,
  shortKAkuznechik = 0x3c,
  longKAmagma = 0x1d,
  longKAkuznechik = 0x3d
} KeyMechanismType
```

Описатель KeyMechanismType предназначен для указания конкретных значений параметров алгоритмов преобразования ключевой информации и выработки производных ключей, используемых транспортным протоколом. Смысловое описание приведенных констант может быть найдено в приложении Г.

При сериализации объект типа KeyMechanismType представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

B.3.15 Описатель AlertType

```
typedef enum {
  unknownError = 0x1000,
  unsupportedCryptoMechanism = 0x1001,
  wrongPreSharedKey = 0x1002,
  wrongInternalPSKIdentifier = 0x1003,
  wrongIntegrityCode = 0x1004,
  lostIntegrityCode = 0x1005,
  wrongCertificateProcessed = 0x100a,
  wrongCertificateNumber = 0x100b,
  expiredCertificate = 0x100c,
  unsupportedCertificateNumber = 0x100d,
  notValidCertificateNumber = 0x100e,
  wrongCertificateApplication = 0x100f,
  wrongCertificateIssuer = 0x1010,
  unsupportedCertificateIssuer = 0x1011,
  unsupportedCertificateFormat = 0x1012,
  wrongCertificateIntegrityCode = 0x1013,
```

```
  usupportedKeyMechanism = 0x1020,
  unsupportedEllipticCurveID = 0x1031,
  wrongEllipticCurvePoint = 0x1032,
  terminateConnection = 0x1041,
  keyResourceTimeUp = 0x1042
} AlertType
```

Описатель AlertType предназначен для указания конкретных значений кодов ошибок, возникающих при реализации защищенного взаимодействия. Каждый код ошибки представляет собой целое неотрицательное число в интервале от 0 до 65535 = $2^{16} - 1$.

При сериализации объекты типа AlertType преобразуются в последовательность октетов в соответствии с алгоритмом, определенным для типа LengthShortInt.

Указанные выше коды определяют следующие ошибки защищенного взаимодействия:

unknownError — неизвестная ошибка; такой код может отправляться как клиентом, так и сервером в случае возникновения ситуации, не предписанной настоящими рекомендациями;

usupportedCryptoMechanism — использование недопустимого или не поддерживаемого криптографического механизма;

wrongPreSharedKey — использование неверного идентификатора предварительно распределенного ключа аутентификации;

wrongInternalPSKIdentifier — переданный идентификатор выработанного абонентами ключа аутентификации не может быть подтвержден;

wrongIntegrityCode — переданные данные содержат неверный код целостности или имитовставку;

lostIntegrityCode — переданные данные не содержат ожидаемый результат контроля целостности;

wrongCertificateProcessed — неверный тип запроса или указания на использование сертификата ключа проверки электронной подписи;

wrongCertificateNumber — неверный номер сертификата ключа проверки электронной подписи;

expiredCertificate — запрос или указание на использование сертификата с неверным интервалом времени использования;

unsupportedCertificateNumber — неподдерживаемый номер сертификата ключа проверки электронной подписи;

notValidCertificateNumber — недопустимый номер сертификата ключа проверки электронной подписи;

wrongCertificateApplication — неверная область применения сертификата ключа проверки электронной подписи;

wrongCertificateIssuer — запрос или указание на использование сертификата с неизвестным центром сертификации (идентификатором удостоверяющего центра);

unsupportedCertificateIssuer — запрос или указание на использование сертификата с неподдерживаемым центром сертификации (идентификатором удостоверяющего центра);

unsupportedCertificateFormat — запрос или указание на использование сертификата с неизвестным или неподдерживаемым форматом;

wrongCertificateIntegrityCode — запрос или указание на использование сертификата, содержащего неверное значение электронной подписи;

usupportedKeyMechanism — запрос на использование неизвестного или неподдерживаемого набора параметров криптографических механизмов;

unsupportedEllipticCurveID — переданные данные содержат неверный или неподдерживаемый идентификатор эллиптической кривой;

wrongEllipticCurvePoint — переданные данные содержат неверную точку эллиптической кривой;

terminateConnection — абонент неожиданно оборвал защищенное взаимодействие;

keyResourceTimeUp — исчерпан ключевой ресурс, необходимо повторно выполнить протокол выработки ключей.

B.3.16 Описатель KeyType

```
typedef enum {
  unknownKey = 0x00,
  derivativeKey = 0x01,
  ePSKKey = 0x02,
  iPSKKey = 0x03
} KeyType
```

Описатель KeyType предназначен для указания типа ключа, используемого при реализации криптографического механизма. Указанные константы определены в соответствии со значениями, определяемыми значениями типа CryptoMechanism.

При сериализации объект типа KeyType представляется в виде последовательности, состоящей из одного октета, значение которого определяется приведенной выше константой.

B.4 Формат сообщений транспортного протокола

B.4.1 Фрейм — Frame

```
typedef struct {
```

```
  FrameType tag;
  LengthShortInt length;
  FrameNumber number;
  OctetString adata;
  MessageType type;
  LengthShortInt meslen;
  OctetString message;
  OctetString padding;
  IntegrityCode icode;
} Frame
```

Описатель Frame определяет структуру данных, являющуюся основным контейнером для передачи данных по каналам связи. Каждый фрейм предназначен для передачи в точности одного сообщения, формат которого определяется одной из структур, определяемых в В.5.

Процедура вложения сообщения во фрейм описана в 7.4.1. Процедура получения сообщения из фрейма описана в 7.5.

Структура данных Frame состоит из следующих полей:

а) tag — тип фрейма, см. В.3.1, представленный в виде одного октета с фиксированным значением;

б) length — длина всего фрейма (в октетах), представленная в виде целого числа, записываемого двумя октетами, см. В.2.4;

в) number — криптографический номер фрейма, см. В.3.13, который позволяет однозначно связать данный фрейм с набором криптографических ключей, используемых для обеспечения конфиденциальности и целостности передаваемой информации;

г) adata — опциональная дополнительная информация, помещаемая в заголовок фрейма в открытом виде; длина данной информации определяется старшими 6 битами значения поля tag, см. 7.4;

д) type — тип вложенного во фрейм сообщения, см. В.3.7;

е) meslen — длина вложенного во фрейм сообщения (в октетах), представленная в виде целого числа, записываемого двумя октетами, см. В.2.4;

ж) message — вложенное во фрейм сообщение;

и) padding — произвольная последовательность октетов конечной длины (дополнение);

к) icode — код целостности или имитовставка, см. В.3.12.

При сериализации объект типа Frame преобразуется в последовательность октетов путем последовательной конкатенации своих полей, см. рисунок В.2.

$$\text{frame} = \underbrace{\text{tag}||\text{length}||\text{number}||\text{adata}}_{\text{Заголовок}}||\underbrace{\text{type}||\text{meslen}||\text{message}||\text{padding}}_{\text{Тело}}||\text{icode}$$

Рисунок В.2 — Формат фрейма

Для описания взаимосвязи между длинами различных фрагментов фрейма определим следующие величины:

l = Len(frame) — длина всего фрейма;

h — длина заголовка, включающего в себя, при наличии, опциональную дополнительную информацию;

m = Len(meslen) — длина сообщения, вкладываемого во фрейм;

p = Len(padding) — длина дополнения;

i = Len(icode) — длина сериализованного представления типа IntegrityCode, содержащего значение кода целостности, тогда выполнено равенство l = h+3 + m + p + i. Данное равенство позволяет однозначно восстановить длину дополнения при получении фрейма, поскольку

$$p = l - (h+3 + m + i).$$

Сериализация объекта типа Frame выполняется в соответствии со следующими равенствами:

```
frame[0] = tag;
frame[1] = length[0];
frame[2] = length[1];
frame[3] = number[0];
frame[4] = number[1];
frame[5] = number[2];
frame[6] = number[3];
frame[7] = number[4];
frame[8] = adata[0];
.....
frame[h] = type;
frame[h+1] = meslen[0];
frame[h+2] = meslen[1];
```

frame[h+3] = message[0];
.....
frame[m+h+2] = message[m-1];
frame[m+h+3] = padding[0];
.....
frame[m+p+h+2] = padding[p-1];
frame[m+p+h+3] = icode[0];
.....
frame[l-1] = icode[i-1].

Примечание — При передаче по каналу связи первые h октетов фрейма всегда передаются в незашифрованном виде.

В.5 Формат сообщений протоколов сеансового уровня

В.5.1 Сообщение ClientHelloMessage

typedef struct {
  CryptoMechanism algorithm;
  PreSharedKeyID idpsk;
  RandomOctetString random;
  EllipticCurvePoint point;
  LengthOctet countOfExtensions;
} ClientHelloMessage

Описатель ClientHelloMessage вводит структуру данных, определяющую формат сообщения, которым клиент инициирует начало протокола выработки общего ключа. Данное сообщение должно вкладываться в структуру Frame со значением поля type, равным clientHello.

Сообщение ClientHelloMessage должно передаваться в незашифрованном виде, процедура его формирования описывается в 5.7.1.

Структура данных ClientHelloMessage состоит из следующих полей:

а) algorithm — алгоритм, используемый для подтверждения целостности данных, передаваемых в незашифрованном виде. Возможные значения данного поля определяются типом CryptoMechanism.

б) idpsk — идентификатор предварительно распределенного ключа аутентификации ePSK или iPSK. Данный параметр является опциональным — если идентификатор не определен, то должно быть установлено значение

idpsk.present = notPresent.

Примечание — Ключи аутентификации iPSK и ePSK определяются в 5.2.1. Механизмы формирования ключей iPSK и ePSK описываются, соответственно в 6.6 и приложении Б;

в) random — случайная (псевдослучайная) последовательность октетов фиксированной длины;

г) point — точка эллиптической кривой, используемая в протоколе выработки ключей;

д) countOfExtensions — определяемое описателем LengthOctet целое неотрицательное число, указывающее количество расширений, которые будут отправлены клиентом серверу после отправки сообщения ClientHelloMessage.

При сериализации типа данных ClientHelloMessage используется последовательная конкатенация полей структуры:

clienthello = algorithm||idpsk||random||point||countOfExtension.

Обозначим p = Len(point), l = Len(idpsk) — длины соответствующих полей сообщения ClientHelloMessage, тогда

clienthello[0] = algorithm[0];
clienthello[1] = algorithm[1];
clienthello[2] = idpsk[0];
.....
clienthello[l+1] = idpsk [l+1];
clienthello[l+2] = random[0];
.....
clienthello[l+34] = random[31];
clienthello[l+35] = point[0];
.....
clienthello[l+p+34] = point[p-1];
clienthello[l+p+35] = countOfExtension

В.5.2 Сообщение ServerHelloMessage

typedef struct {
  CryptoMechanism algorithm;
  RandomOctetString random;

```
  EllipticCurvePoint point;
  LengthOctet countOfExtensions;
} ServerHelloMessage
```

Описатель ServerHelloMessage вводит структуру данных, определяющую формат сообщения, которым сервер отвечает клиенту на сообщение ClientHelloMessage. Сообщение ServerHelloMessage должно помещаться в структуру Frame со значением поля type, равным serverHello и передаваться в незашифрованном виде. Процедура формирования сообщения ServerHelloMessage описывается в 5.7.2.

Структура данных ServerHelloMessage состоит из следующих полей:

а) algorithm — криптографические механизмы, которые будут использованы для шифрования и имитозащиты сообщений, передаваемых в ходе защищенного взаимодействия. Возможные значения данного поля определяются типом CryptoMechanism;

б) random — случайная (псевдослучайная) последовательность октетов фиксированной длины;

в) point — точка эллиптической кривой, используемая в протоколе выработки ключей;

г) countOfExtensions — определяемое описателем LengthOctet целое неотрицательное число, указывающее количество расширений, которые будут отправлены сервером клиенту после отправки сообщения ServerHelloMessage.

При сериализации типа данных ServerHelloMessage используется последовательная конкатенация полей структуры — переменная типа ServerHelloMessage преобразуется в последовательность октетов следующим образом.

$$serverhello = algorithm||random||point||countOfExtension,$$

или

```
serverhello[0] = algorithm[0];
serverhello[1] = algorithm[1];
serverhello[2] = random[0];
.....
serverhello[33] = random[31];
serverhello[34] = point[0];
.....
serverhello[p + 33] = point[p - 1];
serverhello[p + 34] = countOfExtension,
```

где p = Len(point) — длина сериализованного представления точки эллиптической кривой.

B.5.3 Сообщение VerifyMessage

```
typedef struct {
  IntegrityCode mac;
  IntegrityCode sign;
} VerifyMessage
```

Описатель VerifyMessage вводит структуру данных, определяющую формат сообщения, используемого обеими сторонами в ходе выполнения протокола выработки ключей для верификации абонентов и подтверждения ключей, выработанных в ходе выполнения протокола выработки ключей.

Сообщение должно помещаться в структуру Frame со значением поля type, равным verify, и передаваться по каналам связи в зашифрованном виде.

Структура VerifyMessage состоит из следующих полей:

а) mac — определяемое типом IntegrityCode значение кода целостности, выработанное в соответствии с 5.7.2 или 5.7.3; данное значение является опциональным;

б) sign — определяемое типом IntegrityCode значение электронной подписи, выработанное в соответствии с 5.7.2 или 5.7.3; данное значение является опциональным.

Примечание — Протоколом выработки ключей не допускается одновременное отсутствие значения кода целостности mac и значения электронной подписи sign.

При сериализации типа данных VerifyMessage используется последовательная конкатенация полей структуры — переменная типа VerifyMessage преобразуется в последовательность октетов следующим образом

$$verify = mac \,||\, sign,$$

или

```
verify[0] = mac[0];
.....
verify[m-1] = mac[m-1];
verify[m] = sign[0];
.....
verify[m + s - 1] = sign[s - 1],
```

где m = Len(mac), s = Len(sign) — длины соответствующих сериализованных полей сообщения VerifyMessage.

B.5.4 Сообщение ApplicationDataMessage

```
typedef OctetString ApplicationDataMessage
```

Сообщение ApplicationDataMessage представляет собой последовательность октетов произвольной, быть может нулевой, длины. Формат данных, помещаемых в сообщение ApplicationDataMessage, определяется на прикладном уровне и в настоящих рекомендациях не рассматривается.

Данное сообщение должно помещаться в структуру Frame со значением поля type, равным applicationData.

В.5.5 Сообщение AlertMessage

```
typedef struct {
  AlertType code;
  CryptoMechanism algorithm;
  PresentType present;
  OctetString message;
} AlertMessage
```

Описатель AlertMessage вводит структуру данных, определяющую формат сообщения, используемого обеими сторонами защищенного взаимодействия в случае возникновения ошибки или получения некорректных данных.

Сообщение AlertMessage должно помещаться в структуру Frame со значением поля type, равным alert и может передаваться по каналам связи как в зашифрованном, так и в незашифрованном виде.

Структура данных AlertMessage состоит из следующих полей:

а) code — код ошибки;

б) algorithm — идентификатор криптографического механизма, см. В.3.8, используемый для контроля целостности сообщений, передаваемых в открытом виде; данное значение может отличаться от значения, выбранного клиентом при инициализации протокола выработки общих ключей, см. 5.7.1;

в) present — флаг того, что сообщение содержит дополнительное текстовое сообщение;

г) message — последовательность октетов, которая может содержать произвольное текстовое сообщение; данное поле является опциональным; если поле message присутствует, то поле present должно принимать значение isPresent.

Примечание — Сообщения об ошибках, передаваемые в ходе выполнения протокола выработки ключей, должны передаваться в незашифрованном виде. Поскольку на момент возникновения ошибки абоненты могут не завершить процесс аутентификации и не согласовать ключевую информацию, механизм контроля целостности сообщений об ошибках, передаваемых в открытом виде, должен определяться отправителем сообщения AlertMessage и указываться в поле AlertMessage.algorithm.

После окончания протокола выработки ключей, сообщения об ошибках должны передаваться в зашифрованном виде. В этом случае целостность сообщений AlertMessage должна обеспечиваться криптографическим механизмом, указанным в поле ServerHelloMessage.algorithm, с использованием текущих производных ключей шифрования и имитозащиты.

При сериализации объект типа AlertMessage представляется в виде конкатенации полей структуры, т. е.

$$alert = code||algorithm||present||message$$

или

```
alert[0] = code[0];
alert[1] = code[1];
alert[2] = algorithm[0];
alert[3] = algorithm[1];
alert[4] = notPresent
```

если последовательность октетов message не определена, либо

```
alert[0] = code[0];
alert[1] = code[1];
alert[2] = algorithm[0];
alert[3] = algorithm[1];
alert[4] = isPresent;
alert[5] = message[0];
.....
alert[l + 4] = message[l-1],
```

где l = Len(message).

В.5.6 Сообщение GeneratePSKMessage

```
typedef struct {
  RandomOctetString random;
  PreSharedKeyID id;
} GeneratePSKMessage
```

Описатель GeneratePSKMessage вводит структуру данных, определяющую формат сообщения, используемого обеими сторонами защищенного взаимодействия в протоколе выработки ключа аутентификации iPSK, см. 6.6.

Сообщение GeneratePSKMessage должно помещаться в структуру Frame со значением поля type, равным generatePSK и передаваться по каналам связи в зашифрованном виде.

Структура данных GeneratePSKMessage состоит из следующих полей:

а) random — случайная последовательность октетов длины 32 октета;

б) id — идентификатор выработанного ключа аутентификации iPSK; возвращается инициатору протокола для подтверждения корректности выработанного ключа аутентификации.

При сериализации типа данных GeneratePSKMessage используется последовательная конкатенация полей структуры — переменная типа GeneratePSKMessage преобразуется в последовательность октетов следующим образом:

pskmessage = random || id

или

```
pskmessage[0] = random[0];
.....
pskmessage[31] = random[31];
pskmessage[32] = id[0];
.....
pskmessage[l + 31] = id[l - 1],
```

где l — длина сериализованного представления поля id.

B.6 Формат расширений

B.6.1 Расширение RequestCertificateExtension

```
typedef struct {
  CertificateProcessedType certproctype;
  OctetString identifier;
} RequestCertificateExtension
```

Описатель RequestCertificateExtension вводит структуру данных, определяющую формат расширения, используемого сторонами протокола для запроса используемых сертификатов ключей проверки электронной подписи, подробнее см. 5.4.1.

Расширение RequestCertificateExtension должно помещаться в структуру Frame со значением поля type, равным extensionRequestCertificate. Допускается передача расширения по каналам связи как в зашифрованном, так и в незашифрованном виде.

Структура RequestCertificateExtension состоит из следующих полей:

а) certproctype — способ уточнения параметров запрашиваемого к использованию сертификата ключа проверки электронной подписи; значение данного поля должно определяться значением из множества CertificateProcessedType;

б) identifier — последовательность октетов, определяющая запрашиваемый сертификат ключа проверки электронной подписи.

При сериализации типа данных RequestCertificateExtension используется последовательная конкатенация полей структуры:

reqcertext = certproctype || identifier,

и

```
reqcertext[0] = certproctype;
reqcertext[1] = identifier[0];
.....
reqcertext[l] = identifier[l - 1],
```

где l = Len(identifier) — длина сериализованного представления поля identifier.

B.6.2 Расширение CertificateExtension

```
typedef struct {
  CertificateFormat format;
  Certificate certificate;
} CertificateExtension
```

Описатель CertificateExtension вводит структуру данных, определяющую формат расширения, используемого сторонами протокола для передачи сертификатов ключей проверки электронной подписи, см. подробнее 5.4.1.

Расширение CertificateExtension должно помещаться в структуру Frame со значением поля type, равным extensionCertificate, и передаваться по каналам связи в зашифрованном виде.

Структура CertificateExtension состоит из следующих полей:

а) format — формат передаваемого сертификата ключа проверки электронной подписи, определяется значением CertificateFormat;

б) certificate — последовательность октетов конечной длины, содержащая в себе сертификат ключа проверки электронной подписи.

При сериализации типа данных CertificateExtension используется последовательная конкатенация полей структуры — переменная типа CertificateExtension преобразуется в последовательность октетов следующим образом

certext = format||certificate,

или

certext[0] = format;
certext[1] = certificate[0];
.....
certext[l] = certificate[l-1],

где l = Len(certificate) — длина сериализованного представления поля certificate.

B.6.3 Расширение SetCertificateExtension

typedef RequestCertificateExtension SetCertificateExtension

и

typedef struct{
  CertificateProcessedType certproctype;
  OctetString identifier;
} SetCertificateExtension

Описатель SetCertificateExtension вводит структуру данных, определяющую формат расширения, используемого сторонами протокола для информирования получателя расширения об используемом сертификате ключа проверки электронной подписи отправителя расширения, подробнее см. 5.4.1.

Расширение SetCertificateExtension должно помещаться в структуру Frame со значением поля type, равным extensionSetCertificate. Допускается передача расширения по каналам связи как в зашифрованном, так и в незашифрованном виде.

Структура SetCertificateExtension состоит из следующих полей:

а) certproctype — способ указания параметров запрашиваемого к использованию сертификата ключа проверки электронной подписи; значение данного поля должно определяться значением из множества CertificateProcessedType и принимать значение number или issuer; значение any полагается недопустимым;

б) identifier — последовательность октетов, определяющая запрашиваемый сертификат ключа проверки электронной подписи.

При сериализации типа данных RequestCertificateExtension используется последовательная конкатенация полей структуры:

setcertext = certproctype||identifier,

или

setcertext[0] = certproctype;
setcertext[1] = identifier[0];
.....
setcertext[l] = identifier[1 – 1]

где l = Len(identifier) — длина сериализованного представления поля identifier.

B.6.4 Расширение InformCertificateExtension

typedef OctetString InformCertificateExtension

Описатель InformCertificateExtension вводит тип данных, определяющий формат расширения, используемого сторонами протокола для указания номера используемого отправителем расширения сертификата ключа проверки электронной подписи, подробнее см. 5.4.1.

Тип данных представляет собой последовательность октетов, содержащую номер сертификата ключа проверки электронной подписи.

Расширение InformCertificateExtension должно помещаться в структуру Frame со значением поля type, равным extensionInformCertificate. Допускается передача расширения по каналам связи как в зашифрованном, так и в незашифрованном виде.

B.6.5 Расширение RequestIdentifierExtension

typedef struct {
  RequestType request;
  OctetString identifier;
} RequestIdentifierExtension

Описатель RequestIdentifierExtension вводит структуру данных, определяющую формат расширения, используемого сторонами протокола для информирования и, при необходимости, запроса идентификатора абонента. Данные идентификаторы используются при выработке общей ключевой информации, см. 5.5.

Расширение RequestIdentifierExtension должно помещаться в структуру Frame со значением поля type, равным extensionRequestIdentifier. Допускается передача расширения по каналам связи как в зашифрованном, так и в незашифрованном виде.

Структура RequestIdentifierExtension состоит из следующих полей:

а) request — флаг запроса идентификатора; если значение данного поля равно isRequested, то абонент, получивший данное расширение, должен отправить в ответ расширение RequestIdentifierExtension со значением собственного идентификатора; если расширение отправляется в ответ на запрос RequestIdentifierExtension, либо ответ не требуется, то значение поля request должно полагаться равным notRequested;

б) identifier — последовательность октетов, содержащая идентификатор абонента.

При сериализации типа данных RequestIdentifierExtension используется последовательная конкатенация полей структуры

reqidext = request || identifier

или, обозначается

reqidext[0] = request;

reqidext[1] = identifier[0];

.....

reqidext[1] = identifier[1 – 1],

где l = Len(identifier) — длина сериализованного представления поля identifier.

B.6.6 Расширение KeyMechanismExtension

typedef struct{

KeyMechanismType mechanism;

} KeyMechanismExtension

Описатель KeyMechanismExtension вводит структуру данных, определяющую формат расширения, используемого сторонами протокола для указания криптографических механизмов выработки производных ключей.

Расширение KeyMechanismExtension должно помещаться в структуру Frame со значением поля type, равным extensionKeyMechanism. Допускается передача расширения по каналам связи как в зашифрованном, так и в незашифрованном виде. Случаи, в которых должно применяться расширение KeyMechanismExtension, рассматриваются в 5.8.6.

При сериализации переменная типа данных KeyMechanismExtension представляется в виде одного октета, значение которого совпадает с константой из множества KeyMechanismType.

Приложение Г
(справочное)

Рекомендуемые значения параметров защищенного взаимодействия

В настоящем приложении приведены рекомендуемые значения параметров защищенного взаимодействия, которые позволяют обеспечить гибкую настройку СКЗИ в зависимости от области их применения.

Регулируемые государством области применения СКЗИ, в соответствии с приказом [5], определяются Положением о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации (Положение ПКЗ—2005).

Для СКЗИ, попадающих под действие Положения ПКЗ—2005, рекомендациями Р 1323565.1.012—2017 вводится классификация СКЗИ. На СКЗИ, не попадающие под действие Положения ПКЗ—2005, регулирование и классификация не распространяются. Далее приведены значения параметров защищенного взаимодействия для СКЗИ как попадающих, так и не попадающих под действие Положения ПКЗ—2005.

Определяем значения следующих параметров:

maxFrameCount — максимально допустимое количество фреймов, конфиденциальность и целостность которых обеспечивается одной парой производных ключей шифрования и выработки имитовставки, см. 7.2;

maxFrameKeysCount — максимально допустимое количество пар производных ключей шифрования $eSFK_{n,m}$, $eCFK_{n,m}$ и производных ключей выработки имитовставки $iSFK_{n,m}$, $iCFK_{n,m}$ для одного фиксированного состояния ключевой информации $CATS_n$ и $SATS_n$, см. 6.3;

maxApplicationSecretCount — максимально возможное количество преобразований ключевой информации $CATS_n$ и $SATS_n$, допустимое в рамках одного сеанса защищенного взаимодействия, см. 6.3.

Все указанные параметры должны принимать натуральные значения и зависеть от используемого при защищенном взаимодействии алгоритма блочного шифрования, а также от размера фрейма, определяемого параметром maxFrameLength, см. 7.2.

а) Для СКЗИ, не попадающих под действие Положения ПКЗ—2005, могут быть использованы следующие значения параметров

maxFrameLength ≤ 16384	«Магма»	«Кузнечик»
maxFrameCount	2^{13}	2^{16}
maxFrameKeysCount	$2^{16} - 1$	$2^{16} - 1$
maxApplicationSecretCount	$2^{8} - 1$	$2^{8} - 1$

Данный набор параметров обозначается значениями baseKeyMechanismMagma и baseKeyMechanismKuznechik перечисления KeyMechanismType для алгоритма блочного шифрования «Магма» и, соответственно алгоритма «Кузнечик».

б) Для СКЗИ, попадающих под действие Положения ПКЗ—2005 и относящихся к классам КС1, КС2, КС3, могут быть использованы следующие значения параметров

maxFrameLength ≤ 1500	«Магма»	«Кузнечик»
maxFrameCount	2^{11}	2^{16}
maxFrameKeysCount	$2^{16} - 1$	$2^{16} - 1$
maxApplicationSecretCount	$2^{8} - 1$	$2^{8} - 1$

Данный набор параметров обозначается значениями shortKCmagma и shortKCkuznechik перечисления KeyMechanismType для алгоритма блочного шифрования «Магма» и, соответственно алгоритма «Кузнечик».

maxFrameLength ≤ 16384	«Магма»	«Кузнечик»
maxFrameCount	2^{8}	2^{12}
maxFrameKeysCount	$2^{16} - 1$	$2^{16} - 1$
maxApplicationSecretCount	$2^{16} - 1$	$2^{8} - 1$

Данный набор параметров обозначается значениями longKCmagma и longKCkuznechik перечисления KeyMechanismType для алгоритма блочного шифрования «Магма» и, соответственно алгоритма «Кузнечик».

в) Для СКЗИ, попадающих под действие Положения ПКЗ — 2005 и относящихся к классам КВ, КА, могут быть использованы следующие значения параметров

maxFrameLength ≤ 1500	«Магма»	«Кузнечик»
maxFrameCount	2^5	2^8
maxFrameKeysCount	$2^{16} - 1$	$2^{16} - 1$
maxApplicationSecretCount	$2^{16} - 1$	$2^{16} - 1$

Данный набор параметров обозначается значениями shortKAmagma и shortKAkuznechik перечисления KeyMechanismType для алгоритма блочного шифрования «Магма» и, соответственно, алгоритма «Кузнечик».

maxFrameLength ≤ 16384	«Магма»	«Кузнечик»
maxFrameCount	2^2	2^6
maxFrameKeysCount	$2^{16} - 1$	$2^{16} - 1$
maxApplicationSecretCount	$2^{16} - 1$	$2^{16} - 1$

Данный набор параметров обозначается значениями longKAmagma и longKAkuznechik перечисления KeyMechanismType для алгоритма блочного шифрования «Магма» и, соответственно алгоритма «Кузнечик».

Библиография

[1] Федеральный закон от 6 апреля 2011 г. № 63-ФЗ «Об электронной подписи»
[2] RFC 5869 Krawczyk H. HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
[3] Blom R. Nonpublic key distribution//Advances in Cryptology. — Proceedings of EUROCRYPT'82.—1983.—pp. 231—236.
[4] ISO/IEC 9899:2018 Информационная технология. Языки программирования (Information technology — Programming languages)
[5] Приказ ФСБ России от 9 февраля 2005 г. № 66 «Об утверждении Положения о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации (Положение ПКЗ—2005)»

УДК 681.3.06:006.354 ОКС 35.040

Ключевые слова: информационная технология, криптографическая защита информации, защищенное взаимодействие, контрольные и измерительные устройства

БЗ 1—2020/66

Редактор *Н.А. Аргунова*
Технический редактор *И.Е. Черепкова*
Корректор *М.В. Бучная*
Компьютерная верстка *Е.О. Асташина*

Сдано в набор 14.01.2020. Подписано в печать 10.06.2020. Формат 60×84 1/8. Гарнитура Ариал.
Усл. печ. л. 7,44. Уч.-изд. л. 5,95.
Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Создано в единичном исполнении во ФГУП «СТАНДАРТИНФОРМ» для комплектования Федерального информационного фонда стандартов, 117418 Москва, Нахимовский пр-т, д. 31, к. 2.
www.gostinfo.ru info@gostinfo.ru