МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

Информационная технология

КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ

Криптографические механизмы защищенного взаимодействия контрольных и измерительных устройств

дата введения – 20 – . – . –

1 Область применения

Описываемые в настоящем документе механизмы могут быть отнесены к сеансовому и транспортному уровням модели взаимосвязи открытых систем (далее – ВОС) согласно ГОСТ Р ИСО/МЭК 7498-1-99 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[1]:

- 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.

[1] При переводе на английский язык рекомендуется использовать следующие названия протоколов — Keys Generation Protocol и Application Data Transmission Protocol.

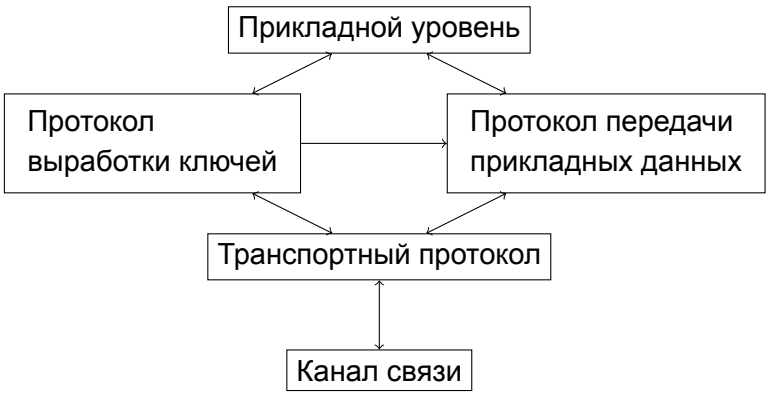

Рисунок 1. Схема информационного обмена.

Описываемые в настоящем документе механизмы могут применяться в средствах криптографической защиты информации (далее – СКЗИ) всех классов, определяемых рекомендациями [1].

Соответствие изложенным в [1] принципам позволяет реализовать механизм регулярного изменения ключевой информации, используемой для обеспечения целостности и конфиденциальности передаваемой информации.

Изменение ключевой информации в ходе одного сеанса связи позволяет передавать очень 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. Численные значения максимального объема 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 7498-1-99 Информационная технология. Взаимосвязь открытых систем. Базовая эталонная модель. Часть 1. Базовая модель.

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 проверяется корректно, 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 **октет:** символ, который может быть представлен в виде двоичной последовательности длины восемь.

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

3.6 **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.

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

3.8 **сериализация:** процесс перевода структуры данных в последовательность октетов конечной длины.

4 Обозначения

В настоящих рекомендациях использованы следующие обозначения:

$\mathbb{B}^*$	множество всех последовательностей октетов конечной длины, включая последовательность нулевой длины;
$\mathbb{B}^s$	множество всех последовательностей октетов, состоящих в точности из s октетов;
$\mathbb{V}^*$	множество всех двоичных последовательностей конечной длины, включая последовательность нулевой длины;
$\mathbb{V}^s$	множество всех двоичных последовательностей, состоящих в точности из s октетов;
$\mathtt{Len}(o)$	функция, возвращающая в качестве значения длину последовательности октетов $o \in \mathbb{B}^*$;
$\mathtt{Ser}(t)$	последовательность октетов конечной длины, являющаяся результатом сериализации одной из структур данных, определяемых в приложении В;
$\mathtt{Msb}_n(o)$	функция, возвращающая подпоследовательность октетов, состоящую из n октетов со старшими номерами последовательности октетов $o \in \mathbb{B}^*$;
p, q	простые числа;

$\mathbb{F}_p$	конечное простое поле, над которым определена эллиптическая кривая E, используемая при реализации защищенного взаимодействия;
$\mathbb{F}_q^*$	мультипликативная группа конечного простого поля $\mathbb{F}_q$;
$\mathbb{F}_{2^n}[x]$	кольцо многочленов от одной переменной, коэффициенты которых принадлежат конечному полю $\mathbb{F}_{2^n}$;
$\mathbb{F}_{2^n}[x, y]$	кольцо многочленов от двух переменных, коэффициенты которых принадлежат конечному полю $\mathbb{F}_{2^n}$;
E	эллиптическая кривая, определенная над полем $\mathbb{F}_p$ либо в канонической форме Вейерштрасса $y^2 \equiv x^3 + ax + b \pmod p$, либо в форме скрученной кривой Эдвардса $eu^2 + v^2 \equiv 1 + du^2v^2 \pmod p$, см. [2];
a, b	элементы поля $\mathbb{F}_p$, являющиеся коэффициентами эллиптической кривой, заданной в канонической форме Вейерштрасса;
e, d	элементы поля $\mathbb{F}_p$, являющиеся коэффициентами эллиптической кривой, заданной в форме скрученной кривой Эдвардса;
$\mathcal{O}$	нейтральный элемент группы точек эллиптической кривой E (бесконечно удаленная точка);
P, Q	точки эллиптической кривой E, определяемые парой координат (элементов поля $\mathbb{F}_p$) — 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 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 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_c, ID_s	последовательности октетов, определяющие идентификаторы, соответственно, клиента и сервера;
$\mathtt{E}(K, M)$	алгоритм зашифрования одного блока информации M на ключе K с помощью алгоритма блочного шифрования; перечень допустимых алгоритмов блочного шифрования определяется в B.2.8;
$\mathtt{Enc}(K, M, I)$	алгоритм зашифрования сообщения M произвольной длины на ключе K с использованием синхропосылки I; перечень допустимых алгоритмов зашифрования (режимов работы блочных шифров) определяется в B.2.8;
$\mathtt{Dec}(K, M)$	алгоритм расшифрования сообщения M произвольной длины с помощью ключа K; перечень допустимых алгоритмов расшифрования (режимов работы блочных шифров) определяется в B.2.8;
$\{M\}_K$	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;

$\texttt{Streebog}_n$ регламентируемая стандартом ГОСТ Р 34.11-2012 бесключевая функции хэширования «Стрибог» с длиной блока n бит, где величина $n \in \{256, 512\}$,

$\lfloor M \rfloor$ обозначение результата применения произвольной бесключевой функции хэширования к сообщению M, используемое на содержащихся в настоящих рекомендациях рисунках;

$\texttt{HMAC}_n$ алгоритм выработки имитовставки HMAC с длиной кода n бит, регламентируемый рекомендациями [6];

$\texttt{Mac}(K, M, I)$ алгоритм выработки имитовставки под сообщением M на ключе K с использованием синхропосылки I (использование синхропосылки является опциональным); перечень допустимых алгоритмов выработки имитовставки определяется в разделе B.2.8;

$\lfloor M \rfloor_K$ обозначение алгоритма выработки имитовставки под сообщением M на ключе K, используемое на содержащихся в настоящих рекомендациях рисунках;

$\texttt{Cert}_c, \texttt{Cert}_s$ сертификат ключа проверки электронной подписи клиента и, соответственно, сервера;

$\texttt{Sign}(d, M)$ алгоритм выработки электронной подписи под сообщением M с помощью ключа электронной подписи d; перечень допустимых алгоритмов выработки электронной подписи определяется в разделе B.2.8;

$\texttt{Ver}(Q, M, s)$ алгоритм проверки электронной подписи s под сообщением M с помощью ключа проверки электронной подписи Q; перечень допустимых алгоритмов проверки электронной подписи определяется в разделе B.2.8;

$\texttt{True}, \texttt{False}$ булевы переменные, принимающее значения, соответственно, «истина» и «ложь», и являющиеся результатом проверки электронной подписи и/или проверки совпадения имитовставки.

$\texttt{Validate}(x)$ функция, проверяющая действительность сертификата x ключа проверки электронной подписи и возвращающая «истину» в случае действительности сертификата и «ложь» в противном случае;

$F[n]$ операция выбора из последовательности октетов F заданного октета с индексом n;

$F[n, \ldots, m]$ операция выбора из последовательности октетов F подпоследовательности октетов; при этом первый элемент подпоследовательности имеет индекс n, а последний элемент — индекс m.

5 Протокол выработки ключей

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 рекомендациями [9].

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 обмена сообщениями в ходе выполнения протокола выработки ключей может быть 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 B.5.

В случае возникновения ошибки абонент (клиент или сервер), обнаруживший ошибку, отправляет сообщение `AlertMessage`, после чего следует реакция, определяемая порядком поведения абонента при получении того или иного сообщения об ошибке, см. B.4.5.

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, 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.2.9.

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 прикладного протокола передачи данных, см. раздел 6, для обеспечения конфиденциальности и целостности передаваемой информации.

5.1 Ключевая система

Ключевая система протокола выработки ключей состоит из следующего множества:

- ключей аутентификации, предназначенные 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 прикладного протокола передачи данных для выработки производных ключей шифрования и выработки имитовставки.

5.1.1 Ключи аутентификации

В протоколе выработки ключей могут быть использованы следующие ключи аутентификации:

- $d_c \in \mathbb{F}_q^*$ – ключ электронной подписи клиента, удовлетворяющий национальному стандарту ГОСТ Р 34.10-2012;
- $Q_c \in E$ – ключ проверки электронной подписи клиента, математически связанный с ключом электронной подписи d_c и удовлетворяющий национальному стандарту ГОСТ Р 34.10-2012; корректность ключа проверки электронной подписи клиента и его однозначное соответствие идентификатору клиента ID_c должны гарантироваться сертификатом $Cert_c$ ключа проверки электронной подписи;
- $d_s \in \mathbb{F}_q^*$ – ключ электронной подписи сервера, удовлетворяющий национальному стандарту ГОСТ Р 34.10-2012;

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

- $\texttt{ePSK} \in \mathbb{B}_n$ – предварительно распределенный, общий для клиента и сервера ключ аутентификации длиной n октетов, где $n \in \{32, \ldots, 64\}$; ключ ePSK является опциональным;

- $\texttt{iPSK} \in \mathbb{B}_{32}$ – 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 общих ключей.

Алгоритм выработки ключа аутентификации iPSK и связывания данного ключа с идентификатором $ID_{\texttt{iPSK}}$ описывается в 6.5. 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.

5.1.2 Общая ключевая информация

В ходе выполнения протокола клиентом выбирается эллиптическая кривая E из множества эллиптических кривых, идентификаторы которых определены в В.2.9. 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:

$$Q = kP = \underbrace{P + \cdots + P}_{k \text{ раз}},$$

где $k \equiv k_c k_s c \pmod{q}$, а величины $k_c, k_s \in \mathbb{F}_q^*$ – случайные вычеты, вырабатываемые, соответственно, клиентом и сервером независимо друг от друга, см. 5.6.1 и 5.6.2. Требования к механизмам генерации случайных значений k_c, k_s определяются рекомендациями [1, раздел 5].

5.1.3 Ключи шифрования и ключи выработки имитовставки

В ходе выполнения протокола клиентом или сервером вырабатывается ключевая информация[2], предназначенная для для шифрования и контроля целостности сообщений, передаваемых в ходе выполнения протокола выработки ключей:

а) ключевая информация $\mathtt{SHTS} \in \mathbb{B}_{64}$, 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:

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

Указанные ключи определяются равенствами:

$$\mathtt{eSHTK} = \mathtt{SHTS}[0, \ldots, 31], \quad \mathtt{iSHTK} = \mathtt{SHTS}[32, \ldots, 63].$$

б) ключевая информация $\mathtt{CHTS} \in \mathbb{B}_{64}$, 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:

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

Указанные ключи определяются равенствами:

$$\mathtt{eCHTK} = \mathtt{CHTS}[0, \ldots, 31], \quad \mathtt{iCHTK} = \mathtt{CHTS}[32, \ldots, 63].$$

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 разделе 5.5.1.

[2]При переводе на английский язык рекомендуется использовать следующие названия ключевой информации – Server Handshake Traffic Secret и Client Handshake Traffic Secret.

5.1.4 Ключевая информация для протокола передачи прикладных данных

В результате выполнения протокола выработки ключей формируется следующая ключевая информация[3]:

а) ключевая информация $\texttt{SATS} \in \mathbb{B}_{64}$, 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 $\texttt{CATS} \in \mathbb{B}_{64}$, 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 $\texttt{iPSK}$ и/или $\texttt{ePSK}$, а также сообщения, передаваемые в ходе выполнения протокола. Алгоритм выработки указанной ключевой информации описывается в разделе 5.5.2.

5.2 Идентификация абонентов

Клиент и сервер должны обладать собственными идентификаторами — последовательностями октетов произвольной длины, обозначаемыми ID_c и ID_s.

Длины идентификаторов должны быть отличны от нуля и могут принимать произвольные положительные значения. Рекомендуется определять идентификаторы последовательностями октетов, длина которых не менее восьми.

5.2.1 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. [4, 5];

[3]При переводе на английский язык рекомендуется использовать следующие названия ключевой информации – Server Application Traffic Secret и Client Application Traffic Secret.

б) назначены либо 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.

5.2.2 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. 5.6.2 и 5.6.3;

б) путем направления расширения `RequestIdentifierExtension`, содержащего идентификатор абонента, см. 5.7.5;

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.

5.3 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. 5.1.1, 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. 5.6.1.

5.3.1 Аутентификация с использованием ключей проверки 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 5.7.4;

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 действительности сертификата ключа проверки электронной подписи сервера (клиента);
- 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:

в) обмен сертификатами может производиться в ходе выполнения протокола выработки общих ключей; для обмена должен использоваться механизм запросов и ответов, реализуемых путем пересылки между абонентами расширений фиксированного формата, см. В.5;

г) 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:

д) 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. 5.6.1 и см. 5.6.2. В ответ на запрос клиента (сервера) сервер (клиент) должен направить сертификат ключа проверки электронной подписи, используя для этого расширение `CertificateExtension`.

е) 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. 5.6.1 и см. 5.6.2. 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`.

ж) 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. 5.6.1 и см. 5.6.2. 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.

5.3.2 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 либо ключ `ePSK`, либо ключ `iPSK`, выработанный в ходе предыдущего сеанса защищенного взаимодействия, см. 5.1.1.

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; при этом для шифрования используются различные для клиента и сервера производные ключи `eCHTS` и `iSHTS`, см. 5.1.3, выработанные из ключа аутентификации.

5.4 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 $x(Q)$; способ преобразования определен в B.2.10;

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. 5.6.2, то текущее значение последовательности октетов R_2 конкатенируется со значением ID_c, то есть:

$$R_2 = R_2 || ID_c;$$

г) если при создании сообщения `ClientHelloMessage` клиентом был использован идентификатор $ID_{\texttt{iPSK}}$ ключа аутентификации `iPSK`, см. 5.6.1, то текущее значение последовательности октетов R_1 и текущее значение последовательности октетов R_2 конкатенируются со значением ключа `iPSK`, то есть:

$$R_1 = R_1 || \texttt{iPSK}, \quad R_2 = R_2 || \texttt{iPSK};$$

д) если при создании сообщения ClientHelloMessage клиентом был использован идентификатор ID_{ePSK} ключа аутентификации ePSK, см. 5.6.1, то текущее значение последовательности октетов R_1 и текущее значение последовательности октетов R_2 конкатенируются со значением ключа ePSK, то есть:

$$R_1 = R_1||\text{ePSK}, \quad R_2 = R_2||\text{ePSK}.$$

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

$$R_1 = x(Q)||\text{iPSK*}||\text{ePSK*}, \quad R_2 = ID_s||ID_c\text{*}||\text{iPSK*}||\text{ePSK*},$$

где знак «*» означает, что данная последовательности 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, то есть:

   $$H_1 = H_1||\text{ExtensionMessage1}||\cdots||\text{ExtensionMessageNc};$$

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

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.

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, то есть:

   $$H_2 = H_2||\text{ExtensionMessage1}||\cdots||\text{ExtensionMessageNs};$$

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, то есть $H_3 = H_3 || \texttt{VerifyMessage}$.

4. Последовательность октетов H_4 формируется следующим образом:

- последовательность октетов H_4 определяется равенством $H_4 = H_3$;

- если клиентом в ответ на сообщение `ClientHelloMessage` отправлялись расширения `ExtensionMessageC1`, ..., `ExtensionMessageCc`, то текущее значение последовательности октетов H_4 конкатенируется с указанными расширениями в незашифрованном виде, в порядке их отправления, то есть

$$H_4 = H_4 || \texttt{ExtensionMessageC1} || \cdots || \texttt{ExtensionMessageCc};$$

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, то есть $H_5 = H_5 || \texttt{VerifyMessage}$.

Примечания.

1. Последовательности октетов $H_1, \ldots, H_5$ представляют собой конкатенации сообщений и расширений, последовательно передаваемых в ходе выполнения протокола выработки ключей. Схема процесса выработки последовательностей октетов $H_1, \ldots, H_5$ 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.

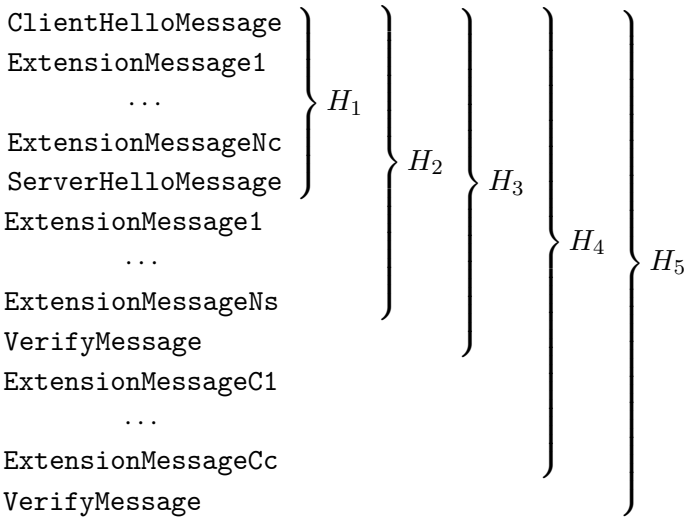

Рисунок 3. Схема формирования последовательностей октетов $H_1, \ldots, H_5$.

5.5 Формирование ключевой информации

В этом разделе описываются алгоритмы формирования ключевой информации, определенной в разделе 5.1.

5.5.1 Формирование ключей шифрования и ключей выработки имитовставки

Ключи шифрования $\texttt{eSHTK}$, $\texttt{eCHTK}$ и ключи выработки имитовставки $\texttt{iSHTK}$, $\texttt{iCHTK}$ 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 $\texttt{eSHTK}$, $\texttt{eCHTK}$ и $\texttt{iSHTK}$, $\texttt{iCHTK}$ используется общая ключевая информация Q, а также определенные в разделе 5.4 последовательности октетов R_1 и H_1, H_3.

Для формирования ключей необходимо выполнить следующую последовательность действий:

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

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

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

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

и определить ключи $\texttt{eSHTS}$ и $\texttt{iSHTS}$ равенствами

$$\texttt{eSHTK} = \texttt{SHTS}[0, \ldots, 31], \quad \texttt{iSHTK} = \texttt{SHTS}[32, \ldots, 63];$$

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

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

и определить ключи $\texttt{eCHTS}$ и $\texttt{iCHTS}$ равенствами

$$\texttt{eCHTK} = \texttt{CHTS}[0, \ldots, 31], \quad \texttt{iCHTK} = \texttt{CHTS}[32, \ldots, 63].$$

Описанная выше последовательность действий схематично изображена на рисунке 4.

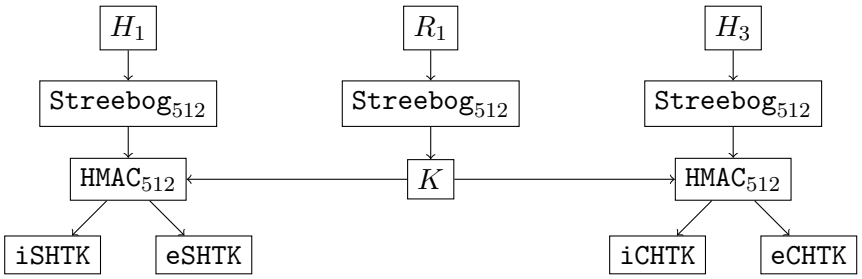

Рисунок 4. Схема выработки ключей для шифрования сообщений в ходе выполнения протокола формирования общей ключевой информации.

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

Ключевая информация $\texttt{CATS}$, $\texttt{SATS}$ является целью выполнения протокола выработки ключей. При формировании указанной ключевой информации используется

общая ключевая информация Q, а также последовательности октетов R_2 и H_5, определенные нами в разделе 5.4. Для формирования ключевой информации $\texttt{CATS}$, $\texttt{SATS}$ необходимо выполнить следующую последовательность действий:

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

$$T = \texttt{HMAC}_{512}(x(Q), R_2);$$

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

$$A_0 = \texttt{Streebog}_{512}(H_5);$$

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

$$A_1 = \texttt{HMAC}_{512}(T, A_0);$$

г) определить последовательность октетов $\texttt{CATS} \in \mathbb{B}_{64}$ равенством:

$$\texttt{CATS} = \texttt{HMAC}_{512}(T, A_1||A_0);$$

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

$$A_2 = \texttt{HMAC}_{512}(T, A_1);$$

е) определить последовательность октетов $\texttt{SATS} \in \mathbb{B}_{64}$ равенством:

$$\texttt{SATS} = \texttt{HMAC}_{512}(T, A_2||A_0).$$

Описанная выше последовательность действий схематично изображена на рисунке 5.

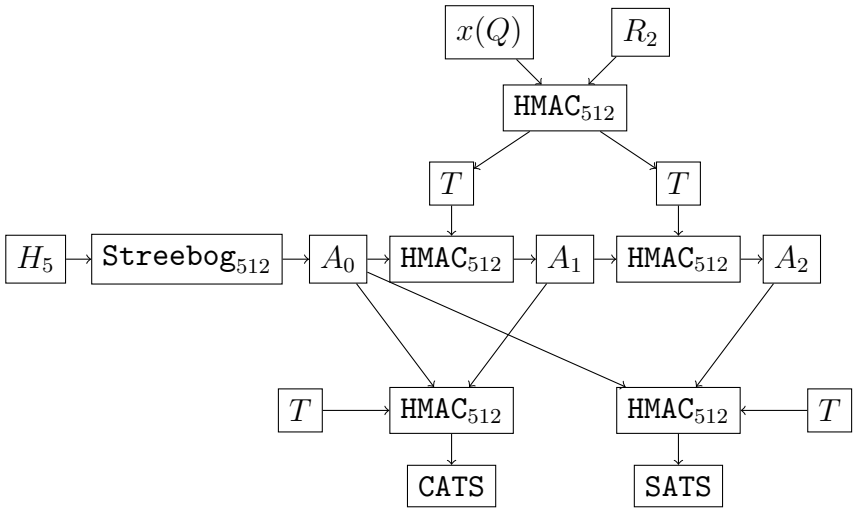

Рисунок 5. Схема выработки ключей для протокола передачи прикладных данных.

5.6 Описание протокола

В этом разделе дается детальное описание последовательности действий, производимых клиентом и сервером в ходе выполнения протокола выработки ключей. Действия, описываемые в разделах 5.6.1 и 5.6.3, выполняются клиентом, действия, описываемые в разделах 5.6.2 и 5.6.4, выполняются сервером.

5.6.1 Формирование сообщения `ClientHelloMessage`

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

Для формирования сообщения `ClientHelloMessage` клиент выполняет следующую последовательность действий.

а) Клиент вырабатывает случайную последовательность октетов длины 32 октета и помещает ее в поле `ClientHelloMessage.random`.

б) Клиент выбирает идентификатор ID_E одной из эллиптических кривых, определяемых типом данных `EllipticCurveID`.

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 длины 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 длины 32 октета. В случае невозможности использования различных генераторов, используемый генератор должен удовлетворять условиям, приведенным в рекомендациях [1, раздел 5.2].

г) Клиент вычисляет точку кривой:

$$P_c = k_c P = \underbrace{P + \cdots + P}_{k_c \text{ раз}};$$

и помещает координаты вычисленной точки в поля структуры `ClientHelloMessage.EllipticCurvePoint`, то есть:

$$\begin{aligned} &\texttt{ClientHelloMessage.point.id} = ID_E;\\ &\texttt{ClientHelloMessage.point.x} = x(P_c);\\ &\texttt{ClientHelloMessage.point.y} = y(P_c). \end{aligned}$$

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

$$\begin{aligned} &\texttt{ClientHelloMessage.idepsk.present} = \texttt{isPresent};\\ &\texttt{ClientHelloMessage.idepsk.length} = \texttt{Len}(ID_{\texttt{ePSK}});\\ &\texttt{ClientHelloMessage.idepsk.id} = ID_{\texttt{ePSK}}, \end{aligned}$$

где $ID_{\texttt{ePSK}}$ идентификатор ключа ePSK. В противном случае клиент определяет:

$$\texttt{ClientHelloMessage.idepsk.present} = \texttt{notPresent}.$$

Примечания.

1. В случае аутентификации по ключу ePSK аутентификация сервера с использованием сертификата ключа проверки электронной подписи, см. 5.1.1, может не производиться.

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:

$$\begin{aligned}&\texttt{ClientHelloMessage.idipsk.present} = \texttt{isPresent};\\&\texttt{ClientHelloMessage.idipsk.length} = \texttt{Len}(ID_{\texttt{iPSK}});\\&\texttt{ClientHelloMessage.idipsk.id} = ID_{\texttt{iPSK}},\end{aligned}$$

где $ID_{\texttt{iPSK}}$ идентификатор ключа iPSK. В противном случае клиент определяет:

$$\texttt{ClientHelloMessage.idipsk.present} = \texttt{notPresent}.$$

Примечания.

1. Использование ключа аутентификации iPSK допускается только после его выработки в ходе предыдущего сеанса защищенного взаимодействия в соответствии с 6.5.

2. В случае аутентификации по ключу iPSK аутентификация сервера с использованием сертификата ключа проверки электронной подписи, см. 5.1.1, может не производиться.

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. Допускается одновременное использование для аутентификации сервера двух ключей аутентификации – iPSK и ePSK.

5. Если клиент не указал ни одного идентификатора предварительно распределенного ключа, то есть одновременно выполнено

$$\begin{aligned}&\texttt{ClientHelloMessage.idepsk.present} = \texttt{notPresent},\\&\texttt{ClientHelloMessage.idipsk.present} = \texttt{notPresent},\end{aligned}$$

то клиент выполняет аутентификацию сервера с использованием сертификата ключа проверки подписи.

ж) Клиент определяет количество расширений Nc, 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 и/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. 5.6.2.

и) В соответствии с описанным в B.4.1 методом созданное сообщение ClientHelloMessage представляется в виде последовательности октетов и передается транспортному протоколу для отправки серверу в незашированном виде.

В случае, если клиентом принято решение об отправке серверу расширений, см. перечисление ж), то клиент переходит к их формированию. Допускается использование 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. 5.6.2. В случае, когда клиентом выбрана аутентификация по предварительно распределенному ключу iPSK и/или ePSK, отправляемое сервером сообщение VerifyMessage всегда содержит код целостности mac и выполнено:

$$\texttt{VerifyMessage.mac.present} = \texttt{isPresent}.$$

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 sign и

$$\texttt{VerifyMessage.sign.present} = \texttt{isPresent}.$$

5.6.2 Формирование сообщения ServerHelloMessage

На втором этапе выполнения протокола выработки общих ключей сервер получает отправленное клиентом сообщение и, при наличии, расширения, выполняет контроль целостности полученных данных, формирует сообщение ServerHelloMessage, а также, при необходимости, формирует 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 оправляет 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.idepsk.present содержит константу isPresent, то выполняются следующие проверки:

   б.1) если идентификатор ключа, содержащийся в поле ClientHelloMessage.idepsk.id, отвергается сервером, то сервер оправляет клиенту сообщение AlertMessage со значением wrongExternalPreSharedKey и завершает сеанс защищенного взаимодействия;

   Примечание. Причины, по которым сервер может отвергнуть ключ аутентификации ePSK, 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.

   б.2) 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; в случае нарушения целостности сервер оправляет клиенту сообщение AlertMessage со значением wrongIntegrityCode и завершает сеанс защищенного взаимодействия; в случае успешной проверки целостности сервер переходит к перечислению д).

в) Если значение поля `ClientHelloMessage.idipsk.present` содержит константу `isPresent`, то выполняются следующие проверки:

в.1) если идентификатор ключа, содержащийся в поле `ClientHelloMessage.idipsk.id`, отвергается сервером, то сервер оправляет клиенту сообщение `AlertMessage` со значением `wrongInternalPreSharedKey` и завершает сеанс защищенного взаимодействия;

в.2) 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 `iPSK`; в случае нарушения целостности сервер оправляет клиенту сообщение `AlertMessage` со значением `wrongIntegrityCode` и завершает сеанс защищенного взаимодействия; в случае успешной проверки целостности сервер переходит к перечислению д).

г) Сервер выполняет контроль целостности сообщения `ClientHelloMessage` с использованием бесключевой функции хеширования; в случае нарушения целостности сервер оправляет клиенту сообщение `AlertMessage` со значением `wrongIntegrityCode` и завершает сеанс защищенного взаимодействия.

д) Используя алгоритм проверки из 5.8.1 сервер проверяет корректность точки эллиптической кривой P_c, содержащейся в структуре `ClientHelloMessage.point`; в случае нарушения корректности точки эллиптической кривой сервер оправляет 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 проверке целостности полученных от клиента расширений, используя при этом алгоритм, определенный значением поля `ClientHelloMessage.algorithm`. В случае, если целостность хотя бы одного расширения нарушена, то сервер оправляет 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 оправляет клиенту сообщение `AlertMessage` с сообщением об ошибке и завершает сеанс защищенного взаимодействия. Коды возможных ошибок приводятся в 5.7.

Примечание. Процедура контроля целостности сообщения `ClientHelloMessage` и последующих за ним расширений может быть реализована на уровне транспортного протокола, см. 7.4.1.

После успешного прохождения всех перечисленных выше проверок сервер формирует сообщение `ServerHelloMessage` и, в случае необходимости, расширения, см. 5.7. Для этого сервер выполняет следующие действия.

з) Сервер вырабатывает случайную последовательность октетов длины 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. 5.6.1, перечисление в), о необходимости использования различных генераторов случайных чисел для выработки значения k_s и случайной последовательности октетов длины 32 октета, также должны применяться при формировании сообщения `ServerHelloMessage`.

к) Сервер вычисляет точку кривой:

$$P_s = k_s P = \underbrace{P + \cdots + P}_{k_s \text{ раз}};$$

и помещает координаты вычисленной точки в поля структуры `ServerHelloMessage.EllipticCurvePoint`, то есть:

$$\begin{aligned} &\texttt{ServerHelloMessage.point.id} = \texttt{ClientHelloMessage.point.id},\\ &\texttt{ServerHelloMessage.point.x} = x(P_s),\\ &\texttt{ServerHelloMessage.point.y} = y(P_s). \end{aligned}$$

л) Сервер определяет количество расширений `Ns`, 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) Сервер вырабатывает ключевую информацию SHTS – для этого сервер формирует определенную в 5.4 последовательность октетов H_1 и применяет определенный в 5.5.1 алгоритм формирования ключевой информации.

п) Если поля iPSK.present и ePSK.present сообщения ClientHelloMessage не содержат значение isPresent, или клиентом направлено одно из расширений RequestCertificateExtension или SetCertificateExtension:

п.1) 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 производится с учетом значений, помещенных клиентом в соответствующее расширение;

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

п.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.

р) 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 используются криптографические механизмы, указанные в поле ServerHelloMessage.algorithm, и выработанная ранее ключевая информация SHTS.

с) В случае необходимости в соответствии с 5.7 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 используются криптографические механизмы, указанные в поле `ServerHelloMessage.algorithm`, и выработанная ранее ключевая информация `SHTS`.

т) Сервер формирует сообщение `VerifyMessage`. Для этого он выполняет следующие действия:

т.1) если ранее, см. перечисление п), сервером определен ключ проверки электронной подписи Q_s, то сервер формирует электронную подпись под последовательностью октетов H_2, см. 5.4, и помещает электронную подпись в сообщение `VerifyMessage`, то есть:

$$\begin{aligned} \texttt{VerifyMessage.sign.present} &= \texttt{isPresent}; \\ \texttt{VerifyMessage.sign.length} &= \texttt{len}; \\ \texttt{VerifyMessage.sign.code} &= \texttt{Sign}(d_s, \texttt{Streebog}_{512}(H_2)), \end{aligned}$$

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:

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

т.2) если клиентом в составе сообщения `ClientHelloMessage` был использован идентификатор предварительно распределенного ключа `iPSK` и/или `ePSK`, то сервер формирует код целостности под последовательностью октетов H_2, см. 5.4, и помещает код целостности в сообщение `VerifyMessage`, то есть:

$$\begin{aligned} \texttt{VerifyMessage.mac.present} &= \texttt{isPresent}; \\ \texttt{VerifyMessage.mac.length} &= 16; \\ \texttt{VerifyMessage.mac.code} &= \texttt{Streebog}_{512}(H_2)[0, \ldots, 15]. \end{aligned}$$

В противном случае сервер определяет значение:

$$\texttt{VerifyMessage.mac.present} = \texttt{notPresent}.$$

у) 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`.

5.6.3 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 оправляет серверу сообщение `AlertMessage` со значением ошибки `unsupportedCryptoMechanism` и завершает сеанс защищенного взаимодействия.

Примечание. Сообщение `AlertMessage` со значением ошибки `unsupported CryptoMechanism` 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 оправляет серверу сообщение `AlertMessage` со значением `wrongIntegrityCode` и завершает сеанс защищенного взаимодействия.

г) Используя алгоритм проверки из 5.8.1 клиент проверяет корректность точки эллиптической кривой P_s, содержащейся в структуре `ServerHelloMessage.point`; в случае нарушения корректности точки эллиптической кривой клиент оправляет серверу сообщение `AlertMessage` со значением `wrongEllipticCurvePoint` и завершает сеанс защищенного взаимодействия.

д) Клиент вырабатывает ключевую информацию `SHTS` – для этого клиент формирует определенную в 5.4 последовательность октетов H_1 и применяет определенный в 5.5.1 алгоритм формирования ключевой информации.

е) 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 оправляет 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. 7.4.1.

ж) 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 оправляет серверу сообщение `AlertMessage` с сообщением об ошибке и завершает выполнение протокола; коды возможных ошибок приводятся в 5.7.

з) 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:

з.1) выбирает соответствующий сертификату ключ проверки электронной подписи Q_s;

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

з.3) с использованием ключа проверки электронной подписи Q_s проверяет истинность подписи под последовательностью октетов H_2, см. 5.4, содержащейся в `VerifyMessage.sign.code`, то есть выполнение тождества:

$$\texttt{Verify}(Q_s, \texttt{Streebog}_{512}(H_2), \texttt{VerifyMessage.sign.code}) = \texttt{True};$$

если тождество не выполнено, то клиент оправляет серверу сообщение `AlertMessage` со значением `wrongIntegrityCode` и завершает сеанс защищенного взаимодействия.

и) Если клиентом в составе сообщения `ClientHelloMessage` был использован идентификатор предварительно распределенного ключа `iPSK` и/или `ePSK`, то клиент:

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

и.2) формирует код целостности под последовательностью октетов H_2, см. 5.4, и проверяет выполнимость равенства:

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

если равенство не выполнено, то клиент оправляет 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 5.4 последовательность октетов H_3 и применяет определенный в 5.5.1 алгоритм формирования ключевой информации.

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

л.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;

л.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.

м.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. 5.4, и помещает электронную подпись в сообщение VerifyMessage, то есть:

$$\begin{aligned} \texttt{VerifyMessage.sign.present} &= \texttt{isPresent}; \\ \texttt{VerifyMessage.sign.length} &= \texttt{len}; \\ \texttt{VerifyMessage.sign.code} &= \texttt{Sign}(d_c, \texttt{Streebog}_{512}(H_4)), \end{aligned}$$

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:

$$\texttt{VerifyMessage.sign.present} = \texttt{notPresent}.$$

м.2) Если клиентом в составе сообщения ClientHelloMessage был использован идентификатор предварительно распределенного ключа iPSK и/или ePSK, или сформированное клиентом значение VerifyMessage.sign.present равно notPresent, то клиент формирует код целостности под последовательностью октетов H_4, см. 5.4, и помещает код целостности в сообщение VerifyMessage, то есть:

$$\begin{aligned}
\texttt{VerifyMessage.mac.present} &= \texttt{isPresent};\\
\texttt{VerifyMessage.mac.length} &= 16;\\
\texttt{VerifyMessage.mac.code} &= \texttt{Streebog}_{512}(H_4)[0,\ldots,15].
\end{aligned}$$

В противном случае сервер определяет значение:

$$\texttt{VerifyMessage.mac.present} = \texttt{notPresent}.$$

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 5.4 последовательность октетов H_5 и применяет определенный в 5.5.2 алгоритм формирования ключевой информации.

После формирования ключевой информации клиент успешно завершает протокол выработки ключей.

5.6.4 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 5.4 последовательность октетов H_3 и применяет определенный в 5.5.1 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 оправляет 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:

г.1) выбирает соответствующий сертификату ключ проверки электронной подписи Q_c;

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

г.3) с использованием ключа проверки электронной подписи Q_c проверяет истинность подписи под последовательностью октетов H_4, см. 5.4, содержащейся в VerifyMessage.sign.code, то есть выполнение равенства:

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

если равенство не выполнено, то сервер оправляет клиенту сообщение AlertMessage с сообщением wrongIntegrityCode и завершает сеанс защищенного взаимодействия.

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

а) проверяет, что поле VerifyMessage.mac.present принимает значение isPresent. Если это не так, сервер оправляет клиенту сообщение AlertMessage с сообщением lostIntegrityCode и завершает сеанс защищенного взаимодействия;

б) формирует код целостности под последовательностью октетов H_4, см. 5.4, и сравнивает ее значение с полученным в составе сообщения VerifyMessage, то есть выполнимость равенства:

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

если равенство не выполнено, то сервер оправляет клиенту сообщение AlertMessage с сообщением wrongIntegrityCode и завершает сеанс защищенного взаимодействия.

е) Сервер формирует ключевую информацию для протокола передачи прикладных данных – для этого сервер формирует определенную в 5.4 последовательность октетов H_5 и применяет определенный в 5.5.2 алгоритм формирования ключевой информации.

После формирования ключевой информации сервер успешно завершает протокол выработки ключей.

5.7 Формирование и проверка корректности расширений

Правила формирования расширений, связанных с идентификацией и аутентификацией абонентов, изложены в 5.2 и 5.3. В этом разделе излагаются механизмы проверки корректности данных расширений, с указанием перечня возникающих ошибок, а также принципы формирования расширений, не связанных с идентификацией и аутентификацией.

5.7.1 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 оправляет второму абоненту сообщение `AlertMessage` со значением `wrongCertificateProcessed`.

б) Если поле `RequestCertificateExtension.certproctype` содержит значение `number`:

б.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 оправляет второму абоненту сообщение `AlertMessage` со значением `wrongCertificateNumber`;

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

б.3) если поле `RequestCertificateExtension.identifier` содержит номер сертификата, которым владеет абонент, то абонент, используя описанный в 5.7.4 алгоритм, проверяет действительность указанного сер-

тификата; в случае нарушения действительности сертификата, абонент принимает решение о некорректности сертификата и оправляет отправителю сообщения 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`:

в.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 оправляет сообщение `AlertMessage` со значением `wrongCertificateIssuer`.

в.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 оправляет 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.

5.7.2 Расширение `SetCertificateExtension`

Расширение `SetCertificateExtension` формируется абонентами с целью 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 должно использоваться 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 оправляет второму абоненту сообщение `AlertMessage` со значением `wrongCertificateProcessed`.

б) Если поле SetCertificateExtension.certproctype содержит значение number:

б.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 оправляет второму абоненту сообщение AlertMessage со значением wrongCertificateNumber;

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

б.3) если поле SetCertificateExtension.identifier содержит номер сертификата, которым владеет абонент, то абонент, используя описанный в 5.7.4 алгоритм, проверяет действительность указанного сертификата; в случае нарушения действительности сертификата, абонент принимает решение о некорректности расширения и оправляет отправителю сообщения сообщение AlertMessage со значением notValidCertificateNumber.

в) Если поле SetCertificateExtension.certproctype содержит значение issuer:

в.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 оправляет сообщение AlertMessage со значением wrongCertificateIssuer.

в.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 оправляет 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.

5.7.3 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.identifier которая не может быть интерпретирована абонентом в качестве номера конкретного сертификата, то абонент принимает решение о некорректности расширения и оправляет 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 оправляет 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 5.7.4, перечисление б), проверяет действительность указанного сертификата; в случае нарушения действительности сертификата, абонент принимает решение о некорректности расширения и оправляет отправителю сообщения 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.

5.7.4 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:

б.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 произведена для каждого сертификата в цепочке. Рекомендуется проводить данную проверку до начала выполнения сеанса защищенного взаимодействия.

б.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`.

б.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`.

б.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.

5.7.5 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 производится, поскольку корректность идентификатора проверяется позднее путем проверки кодов целостности, содержащихся в сообщении `VerifyMessage`.

5.7.6 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 7.2 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 отправить сообщение об ошибке AlertMessage со значением кода ошибки, равным unsupportedKeyMechanism, и завершить сеанс защищенного взаимодействия.

5.8 Вспомогательные алгоритмы

В заключение описываются вспомогательные алгоритмы, которые используются или могут быть использованы клиентом и сервером в ходе выполнения протокола выработки ключей.

5.8.1 Проверка 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.2.9; 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:

б.1) Определить величину x значением поля `EllipticCurvePoint.x`;

б.2) Определить величину y значением поля `EllipticCurvePoint.y`;

б.3) Проверить выполнение сравнения $y^2 \equiv x^3 + ax + b \pmod{p}$, где параметры a, b, p определены идентификатором эллиптической кривой согласно B.2.9; 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:

в.1) Определить величину u значением поля `EllipticCurvePoint.x`;

в.2) Определить величину v значением поля `EllipticCurvePoint.y`;

в.3) Проверить выполнение сравнения $eu^2 + v^2 \equiv 1 + du^2v^2 \pmod{p}$, где параметры e, d, p определены идентификатором эллиптической кривой согласно B.2.9; 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 порядок группы точек m не является простым числом, то необходимо проверить, что точка $[d]P$ отлична от нейтрального элемента группы точек эллиптической кривой (бесконечно удаленной точки $\mathcal{O}$).

Эллиптическая кривая, определяемая значением идентификатора `EllipticCurvePoint.id`, и определенная в перечислении г) точка P должны быть использованы при выработке общей ключевой информации, см. 5.1.2.

6 Протокол передачи прикладных данных

Основная задача протокола передачи прикладных данных заключается 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 указанной информации протоколу транспортного уровня.

Длина формируемых сообщений ограничивается параметром транспортного протокола `maxFrameLength`— максимально допустимой длиной фреймов, в которые вкладываются сформированные сообщения, см. 7.1.

6.1 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

  $$\texttt{SATS}_n, \quad \texttt{CATS}_n,$$

  где n натуральное число, принимающее значения $n = 1, 2, \ldots$; алгоритм преобразования ключевой информации описывается в 6.3;

- 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 $\texttt{SATS}_n$ и обозначаются символами

  $$\texttt{eSFK}_{n,m}, \quad \texttt{iSFK}_{n,m},$$

  где m натуральное число, принимающее значения $m = 1, 2, \ldots$; алгоритм выработки производных ключей сервера описывается в 6.4;

- 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 $\mathtt{CATS}_n$ и обозначаются символами

$$\mathtt{eCFK}_{n,m}, \quad \mathtt{iCFK}_{n,m},$$

где m натуральное число, принимающее значения $m = 1, 2, \ldots$; алгоритм выработки производных ключей клиента описывается в 6.4;

- 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 описывается в 6.5.

6.2 Параметры протокола

Параметрами протокола передачи прикладных данных являются следующие значения:

- maxApplicationSecretCount — максимально возможное количество преобразований ключевой информации $\mathtt{CATS}_n$ и $\mathtt{SATS}_n$, 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 $\mathtt{eSFK}_{n,m}$, $\mathtt{eCFK}_{n,m}$ и производных ключей выработки имитовставки $\mathtt{iSFK}_{n,m}$, $\mathtt{iCFK}_{n,m}$ для одного фиксированного состояния ключевой информации $\mathtt{CATS}_n$ и $\mathtt{SATS}_n$; 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. B.2.8, класса СКЗИ, реализующего настоящие рекомендации по стандартизации, а также параметров транспортного протокола, см. 7.1 Примерные значения указанных выше величин для различных криптографических механизмов приведены в приложении Г.

6.3 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 разделе 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 либо как переменная

типа LengthOctet – в случае, если значение параметра maxApplicationSecretCount удовлетворяет неравенству

$$\texttt{maxApplicationSecretCount} \leq 255,$$

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

$$256 \leq \texttt{maxApplicationSecretCount} \leq 65535.$$

Точный способ представления счетчика n определяется константой типа KeyMechanismType.

Последовательность ключевых значений $\texttt{CATS}_n$ и $\texttt{SATS}_n$ определяется равенствами

$$\begin{aligned}
\texttt{CATS}_1 &= \texttt{CATS}, \\
\texttt{SATS}_1 &= \texttt{SATS}, \\
\texttt{CATS}_{n+1} &= \texttt{HMAC}_{512}(T,\ \texttt{CATS}_n||\texttt{Ser}(n+1)), \\
\texttt{SATS}_{n+1} &= \texttt{HMAC}_{512}(T,\ \texttt{SATS}_n||\texttt{Ser}(n+1)), \quad n = 1, \ldots, \texttt{maxApplicationSecretCount},
\end{aligned}$$

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. 5.5.2. При этом длина последовательности октетов $\texttt{Ser}(n)$ должна совпадать с длиной последовательности октетов, представляющей значение n, и может принимать значение либо 1, либо 2.

Примечания

1. Указанная выше последовательность действий схематично изображена на рисунке 6.

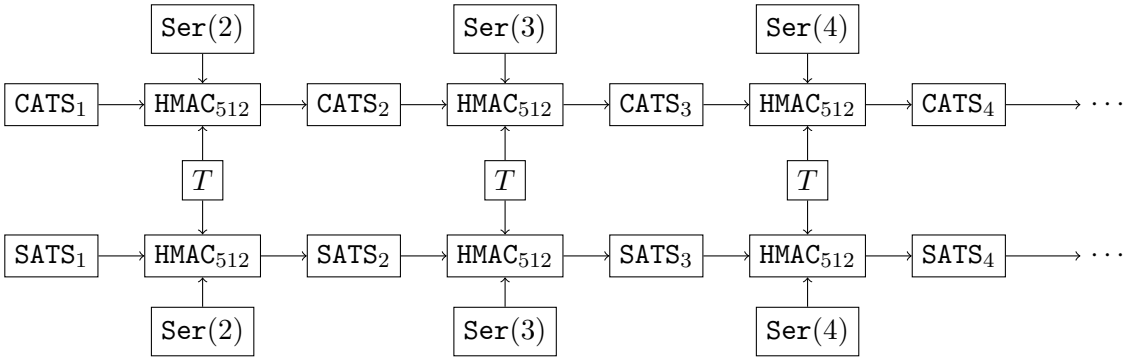

Рисунок 6. Схема преобразования ключевой информации.

2. Указанный способ преобразования ключевой информации, по сути, представляет собой описанный в [11] алгоритм HKDF-Expand на основе отечественной функции хэширования Streebog, независимо примененный к ключевой информации CATS и SATS. Отличия от [11] заключаются в способе определения начального значения и использовании состоящего из двух октетов счетчика ключей.

6.4 Алгоритмы выработки производных ключей

Производные ключи вырабатываются из ключевой информации $\mathtt{CATS}_n$ и $\mathtt{SATS}_n$ для каждого значения счетчика n 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 либо как переменная типа `LengthOctet` – в случае, если значение параметра `maxFrameKeysCount` удовлетворяет неравенству

$$\mathtt{maxFrameKeysCount} \le 255,$$

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

$$256 \le \mathtt{maxFrameKeysCount} \le 65535.$$

Точный способ представления счетчика m определяется константой типа `KeyMechanismType`.

Алгоритм выработки производных ключей шифрования $\mathtt{eCFK}_{n,m}$, $\mathtt{eSFK}_{n,m}$ зависит от алгоритма блочного шифрования, определяемого значением поля `ServerHelloMessage.algorithm`, см. 5.6.2, и определяется следующими равенствами

$$\begin{aligned}
\mathtt{eCFK}_{n,0} &= \mathtt{CATS}_n[0,\dots,31],\\
\mathtt{eCFK}_{n,m} &= ACPKM(\mathtt{eCFK}_{n,m-1}), \quad m = 1,\dots,\mathtt{maxFrameKeysCount},
\end{aligned}$$

$$\begin{aligned}
\mathtt{eSFK}_{n,0} &= \mathtt{SATS}_n[0,\dots,31],\\
\mathtt{eSFK}_{n,m} &= ACPKM(\mathtt{eSFK}_{n,m-1}), \quad m = 1,\dots,\mathtt{maxFrameKeysCount},
\end{aligned}$$

где $ACPKM$ — алгоритм преобразования ключа, регламентируемый рекомендациями [12, раздел 4.1.1] 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

$$\mathtt{K}_m = ACPKM(\mathtt{K}_{m-1}) = \mathtt{E}(\mathtt{K}_{m-1}, D_1)||\cdots||\mathtt{E}(\mathtt{K}_{m-1}, D_J),$$

где $\mathtt{K}_m$ ключ, вырабатываемый из ключа $\mathtt{K}_{m-1}$, а $D_1||\cdots||D_J = D \in \mathbb{B}_{32}$ — константная последовательность октетов, определяемая в рекомендациях [12] следующим образом:

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

и

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

Алгоритм выработки ключей выработки имитовставки $\mathtt{iCFK}_{n,m}$, $\mathtt{iSFK}_{n,m}$ соответственно, клиента и сервера, также зависит от алгоритма блочного шифрования, определяемого значением поля `ServerHelloMessage.algorithm`, см. 5.6.2, и определяется следующим образом.

Для алгоритма блочного шифрования «Магма» значение натурального числа $\mathtt{Ctr}$ определяется равенством

$$\mathtt{Ctr} = 18446744069414584320_{10} = \mathtt{FFFFFFFF00000000}_{16},$$

а для алгоритма «Кузнечик» — равенством

$$\begin{aligned}\mathtt{Ctr} &= 340282366920938463444927863358058659840_{10}\\ &= \mathtt{FFFFFFFFFFFFFFFF0000000000000000}_{16}.\end{aligned}$$

Тогда

$$\begin{aligned}\mathtt{CK}_n &= \mathtt{CATS}_n[32,\ldots,63],\\ \mathtt{SK}_n &= \mathtt{SATS}_n[32,\ldots,63],\\ \mathtt{iCFK}_{n,m} &= \mathtt{E}(\mathtt{CK}_n,\mathtt{Ser}(\mathtt{Ctr}+mJ))||\cdots||\mathtt{E}(\mathtt{CK}_n,\mathtt{Ser}(\mathtt{Ctr}+(m+1)J-1)),\\ \mathtt{iSFK}_{n,m} &= \mathtt{E}(\mathtt{SK}_n,\mathtt{Ser}(\mathtt{Ctr}+mJ))||\cdots||\mathtt{E}(\mathtt{CK}_n,\mathtt{Ser}(\mathtt{Ctr}+(m+1)J-1)),\end{aligned}$$

где величина J определена ранее в перечислении а).

Примечание. Описанные выше алгоритмы выработки ключей клиента – производных ключей шифрования $\mathtt{eCFK}_{n,m}$ и ключей выработки имитовставки $\mathtt{iCFK}_{n,m}$ схематично изображены на рисунке 7. Схема выработки ключей сервера – производных ключей шифрования $\mathtt{eSFK}_{n,m}$ и ключей выработки имитовставки $\mathtt{iSFK}_{n,m}$ аналогична изображенной на рисунке 7.

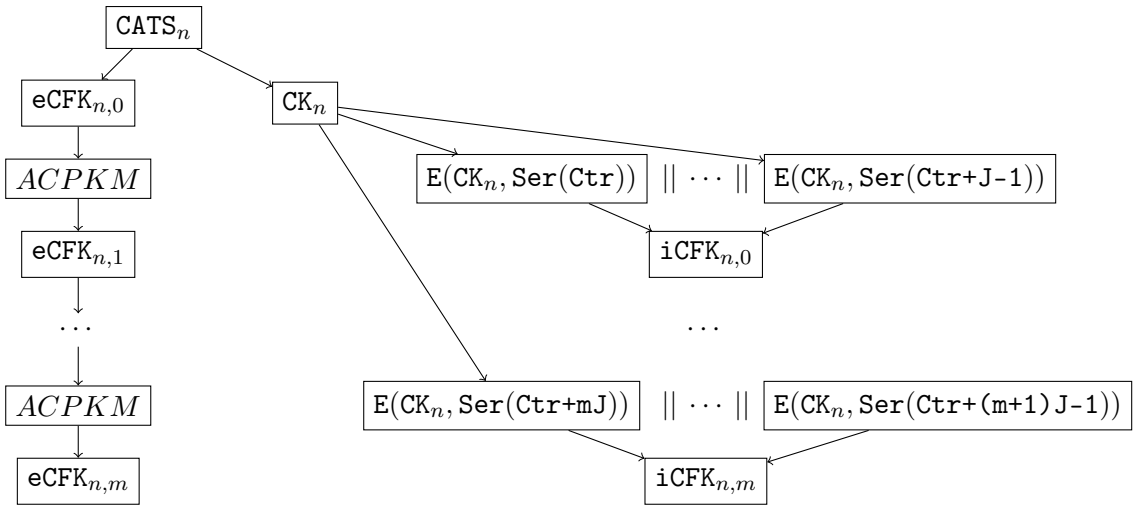

Рисунок 7. Схема выработки производных ключей.

6.5 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 $\mathtt{Rand}_1$ длины 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 $\mathtt{eCFK}_{n,m}$ и $\mathtt{iCFK}_{n,m}$ и текущие значения счетчиков n, m.

После получения сообщения GeneratePSKMessage сервер выполняет следующие действия:

в) вырабатывает случайную последовательность октетов $\mathtt{Rand}_2$ длины 32 октета и помещает ее в поле GeneratePSKMessage.random;

г) вырабатывает последовательность октетов PSK, определяемую равенством

$$\mathtt{PSK} = \mathtt{HMAC}_{512}(T, \mathtt{Rand}_1||\mathtt{Rand}_2||ID_s||ID_c*||\mathtt{FrameNumber}_s||\mathtt{FrameNumber}_c),$$

где

- ID_s, ID_c — идентификаторы сервера и клиента (опционально), использованные клиентом и сервером в процессе выполнения протокола выработки ключей;
- T — общая для клиента и сервера ключевая информация, используемая для преобразования ключевой информации, см. 6.3;
- $\mathtt{FrameNumber}_s$, $\mathtt{FrameNumber}_c$ — 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

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

е) определяет ключ аутентификации iPSK равенством

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

ж) формирует поле GeneratePSKMessage.id равенствами

$$\begin{aligned} \texttt{GeneratePSKMessage.id.present} &= \texttt{isPresent}, \\ \texttt{GeneratePSKMessage.id.length} &= 32, \\ \texttt{GeneratePSKMessage.id.id} &= ID_{\texttt{iPSK}}. \end{aligned}$$

Сформированное сообщение передается сервером транспортному протоколу для отправки в канал связи в зашифрованном виде. При этом, для зашифрования сообщения GeneratePSKMessage и контроля его целостности используются текущие производные ключи $\texttt{eSFK}_{n,m}$ и $\texttt{iSFK}_{n,m}$ и значения счетчиков n, m, использованные при формировании уникального номера $\texttt{FrameNumber}_s$.

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

$$\texttt{PSK} = \texttt{HMAC}_{512}(T, \texttt{Rand}_1||\texttt{Rand}_2||ID_s||ID_c*||\texttt{FrameNumber}_s||\texttt{FrameNumber}_c),$$

где

- ID_s, ID_c — идентификаторы сервера и клиента (опционально), использованные клиентом и сервером в процессе выполнения протокола выработки ключей;
- T — общая для клиента и сервера ключевая информация, используемая для преобразования ключевой информации, см. 6.3;
- $\texttt{FrameNumber}_s$, $\texttt{FrameNumber}_c$ — уникальные номера фреймов, в которых клиент и сервер отправляли сообщения GeneratePSKMessage;

и) определяет идентификатор выработанного ключа iPSK равенством

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

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

$$\texttt{iPSK} = PSK[32, \ldots, 63]$$

и завершает выполнение протокола выработки ключа аутентификации iPSK.

7 Протокол транспортного уровня

Протокол транспортного уровня[4] 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. B.3.1, 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 под собой, 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. B.2.13, 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.

[4] При переводе на английский язык рекомендуется использовать следующее название протокола – Transport Layer Protocol.

Детальное рассмотрение вопросов формирования уникальных номеров фреймов и механизмов связи уникальных номеров фреймов с криптографическими ключами содержатся в 7.1 и 7.2.

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 реализовываться протоколом 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 рекомендациями по стандартизации отдельно для каждого типового механизма реализации канала связи.

7.1 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

$$\texttt{maxFrameLength} \cdot \texttt{maxFrameCount} \leq B_{max},$$

где B_{max} максимально возможный размер последовательности октетов, которая может быть зашифрована на одном ключе. Данное значение зависит от используемых криптографических механизмов, см. B.2.8, класса СКЗИ, реализующего настоящие рекомендации по стандартизации, и не может превосходить величины, определенной в рекомендациях [10, раздел 4]. Примерные значения указанных величин для различных криптографических механизмов приведены в приложении Г.

7.2 Алгоритм формирования уникальных номеров фреймов

Уникальный номер фрейма, представляющий собой последовательность из пяти октетов, см. B.2.13, 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 $\texttt{maxFrameCount} - 1$, 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 зашифрованного 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. 5.6.1 и 5.6.2, 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. Значение 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.

г) Если текущий используемый криптографический механизм определяется константой `standard122`, то счетчики l, m есть переменные типа `LengthShortInt`,

а счетчик n есть переменная типа LengthOctet. В этом случае 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].
```

д) Если текущий используемый криптографический механизм определяется константой standard221, то счетчик l, m есть переменная типа LengthOctet, а счетчики m, n есть переменные типа LengthShortInt. В этом случае 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;
```

7.3 Алгоритм формирования фрейма

Алгоритм формирования фрейма, см. B.3.1, принимает на вход от протокола сеансового уровня следующие параметры:

а) сформированное на сеансовом уровне сообщение или расширение message, представленное в виде последовательности октетов (сериализованного представления одной из определенных в B.4 и B.5 структур данных);

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

в) длину 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) пару ключей — ключ шифрования eK $\in \mathbb{B}^{32}$ и ключ выработки имитовставки iK $\in \mathbb{B}^{32}$, при необходимости;

7.3.1 Процедура вложения

После проверки перечисленных выше параметров на корректность[5], выполняется следующая последовательность действий.

[5]Под корректностью параметров подразумевается факт того, что данные параметры определены некоторыми значениями, принадлежащими области допустимых значений.

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

а.1) поле `tag` полагается равным `plainFrame`, для фреймов, передаваемых в незашифрованном виде, или `encryptedFrame`, для фреймов, передаваемых в зашифрованном виде; необходимость того, должен ли быть зашифрован фрейм, определяется идентификатором криптографического механизма, см. B.2.8, и значениями счетчиков n, m, см. 7.2;

Примечание – согласно 5.6.1 и 5.6.2 в незашифрованном виде могут передаваться сообщения, формируемые в ходе выполнения первого и второго этапов протокола выработки ключей, а также сообщения об ошибках.

а.2) согласно 7.3.3 формируется дополнение `padding` — последовательность октетов длины $\mathtt{Len}(\mathtt{padding})$.

а.3) поле `length` полагается равным

$$11 + \mathtt{Len}(\mathtt{message}) + \mathtt{Len}(\mathtt{padding}) + \mathtt{Len}(\mathtt{icode}),$$

где `icode` – сериализованное представление кода целостности, см. B.2.12. Длина кода целостности однозначно определяется значением используемого идентификатора криптографического механизма.

Примечание. Константа 11 представляет собой сумму 8 байт заголовка, 1 байта, отводимого под хранение типа сообщения M, и двух байт, отводимых под сохранение длины сообщения M.

а.4) поле `number` полагается равным уникальному номеру фрейма.

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

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

б.2) поле `meslen` полагается равным значению $\mathtt{Len}(\mathtt{message})$;

б.3) в поле `message` помещается собственно передаваемое сообщение;

б.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. B.2.8.

г) Если сообщение должно передаваться в зашифрованном виде, то применяется процедура зашифрования и вычисления кода целостности, описываемая в следующем разделе.

Сформированный фрейм передается в канал связи.

7.3.2 Процедура зашифрования и вычисления имитовставки

Процедура зашифрования сообщения и вычисления кода целостности формируемого фрейма зависит от идентификатор 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 стандартом 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 стандартом 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 должен выступать заголовок фрейма – последовательность октетов длины восемь (начиная с нулевого и заканчивая седьмым октетом).

г) для блочного шифра «Магма» (значение идентификатора – magmaCTRplusOMAC) в качестве синхропосылки должен выступать фрагмент заголовка фрейма – последовательность октетов длины четыре (начиная с четвертого и заканчивая седьмым октетом).

Примечание. В случае использования блочного шифра «Магма» согласно ГОСТ Р 34.13-2015 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 учитывается при выборе значений параметров протоколов 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 шифрования с возможностью одновременной выработки имитовставки (режим AEAD), см. [7], 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 шифрования с возможностью одновременной выработки имитовставки значение имитовставки помещается в поле `icode`;

Способ выработки синхропосылки, необходимой для режима шифрования с возможностью одновременной выработки имитовставки, 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 передаваемым в незашифрованном виде заголовком фрейма – последовательностью октетов длины восемь (начиная с нулевого и заканчивая седьмым октетом); то есть

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

где `frame` это последовательность октетов, являющаяся сериализованным представлением сформированного фрейма, см. В.3.1.

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 передаваемым в незашифрованном виде заголовком фрейма, – последовательностью октетов длины восемь (начиная с нулевого и заканчивая седьмым октетом), а оставшиеся октеты принимают нулевые значения, то есть

```
  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.

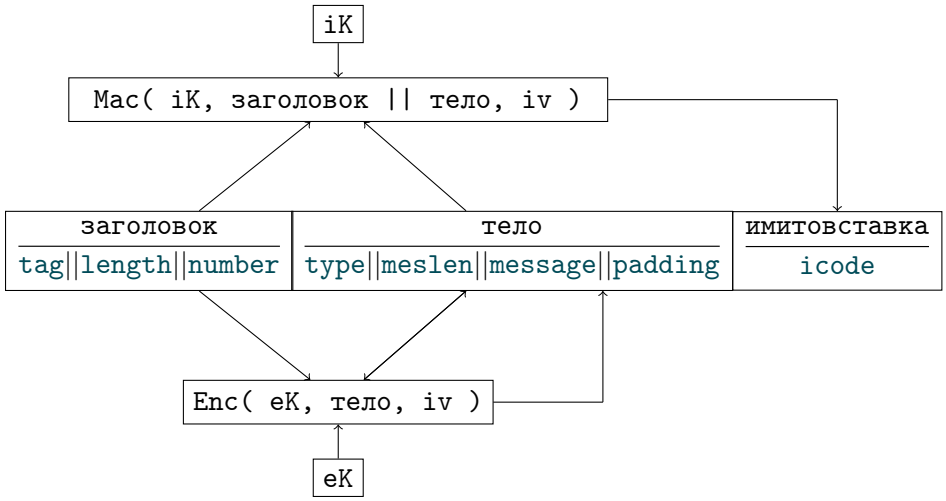

Рисунок 8. Схема зашифрования и вычисления кода целостности фрейма.

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

7.3.3 Процедура генерации дополнения

Дополнение представляет собой последовательность октетов длины $\mathtt{Len}(\mathtt{padding})$. 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. 5.6.1 и 5.6.2. Использование в качестве дополнения последовательности фиксированных значений не рекомендуется.

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

а) длина дополнения может выбираться равной нулю, то есть 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 $\mathtt{Len}(\mathtt{padding})$ должна удовлетворять условию

$$11 + \mathtt{Len}(\mathtt{message}) + \mathtt{Len}(\mathtt{padding}) + \mathtt{Len}(\mathtt{icode}) < \mathtt{maxFrameLength},$$

которое, согласно 7.3.1, следует из определения суммарной длины фрейма.

7.4 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. 5.6.1 и 5.6.2, либо установлено или изменено протоколом сеансового уровня.

7.4.1 Действия при получении незашифрованного фрейма

Согласно описанию протокола выработки общих ключей, см. 5.6, 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

$$\texttt{frame}[0] = \texttt{plainFrame}.$$

Также, согласно 7.2, должно быть проверено выполнение равенств

$$\begin{aligned} \texttt{frame}[3] &= 0;\\ &\dots\\ \texttt{frame}[6] &= 0;\\ \texttt{frame}[7] &= \texttt{number}; \end{aligned}$$

где `number` это уникальное для каждого незашифрованного фрейма натуральное число.

б) Если $\texttt{frame}[7] = 0$, то определить значение идентификатора криптографического механизма, см. B.2.8, используя для этого октеты $\texttt{frame}[11]$ и $\texttt{frame}[12]$. В случае, если $\texttt{frame}[7] > 0$, то использовать идентификатор криптографического алгоритма, определенный ранее.

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

$$\begin{cases} \texttt{frame}[\texttt{idx} - 1] &= \texttt{isPresent},\\ \texttt{frame}[\texttt{idx}] &= \texttt{L}. \end{cases}$$

где $\mathtt{idx} = \mathtt{Len}(\mathtt{frame}) - \mathtt{L}$, а $\mathtt{Len}(\mathtt{frame})$ — длина всего фрейма, определяемая значениями октетов $\mathtt{frame}[1]$ и $\mathtt{frame}[2]$, см. B.3.1.

г) Согласно значению идентификатора криптографического механизма вычислить значение кода целостности или имитовставки от фрагмента фрейма, начинающегося с $\mathtt{frame}[0]$ и заканчивающегося $\mathtt{frame}[\mathtt{idx} - 2]$, где значение idx определено в перечислении в).

Примечания.

1. В случае вычисления имитовставки идентификатор используемого ключа должен быть получен путем разбора вложенного во фрейм сообщения ClientHelloMessage или ServerHelloMessage, см. 5.6.1 и 5.6.2.

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

д) Проверить, что полученное значение кода целостности или имитовставки совпадает со значением, содержащимся во фрагменте полученного фрейма, начинающемся с $\mathtt{frame}[\mathtt{idx} + 1]$ 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 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 $\mathtt{frame}[11]$ сообщение или расширение message, длина которого определяется значением полей $\mathtt{frame}[9]$ и $\mathtt{frame}[10]$, а также тип сообщения или расширения, определяемый значением перечисления MessageType.

7.4.2 Действия при получении зашифрованных фреймов

При получении зашифрованного фрейма должна быть выполнена следующая последовательность действий.

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

$$\mathtt{frame}[0] = \mathtt{encryptedFrame}.$$

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

$$\begin{cases} \mathtt{frame}[\mathtt{idx} - 1] = \mathtt{isPresent}, \\ \mathtt{frame}[\mathtt{idx}] = \mathtt{L}. \end{cases}$$

где $\mathtt{idx} = \mathtt{Len}(\mathtt{frame}) - \mathtt{L}$, а $\mathtt{Len}(\mathtt{frame})$ — длина всего фрейма, определяемая значениями октетов $\mathtt{frame}[1]$ и $\mathtt{frame}[2]$, см. B.3.1.

в) Используя значения полей $\texttt{frame}[3], \ldots, \texttt{frame}[7]$ определить значения счетчиков n, m, см. 7.2, и пару ключей шифрования и выработки имитовставки, однозначно связанных с вычисленными значениями счетчиков.

г) Используя значение идентификатора криптографического механизма, ключи шифрования и выработки имитовставки, а также процедуры, описанные в 7.3.2, расшифровать фрагмент полученного фрейма, начинающийся с $\texttt{frame}[8]$ и заканчивающийся $\texttt{frame}[\texttt{idx}-2]$, где значение idx определено в перечислении б).

д) Вычислить значение имитовставки от фрагмента фрейма, начинающегося с $\texttt{frame}[0]$ и заканчивающегося $\texttt{frame}[\texttt{idx} - 2]$, где значение idx определено в перечислении б).

Примечание. В случае режима шифрования с одновременной выработкой имитовставки, см. В.2.8, перечисления г) и д) должны выполняться одновременно.

е) Проверить, что полученное значение имитовставки совпадает со значением, содержащимся во фрагменте полученного фрейма, начинающемся с $\texttt{frame}[\texttt{idx}+1]$ и заканчивающемся окончанием фрейма, где значение idx определено в перечислении б).

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 проверки успешно пройдены, то протокол транспортного уровня должен передать протоколу сеансового уровня начинающееся с $\texttt{frame}[11]$ сообщение или расширение message, длина которого определяется значением полей $\texttt{frame}[9]$ и $\texttt{frame}[10]$, а также тип сообщения или расширения, определяемый значением перечисления MessageType.

А Приложение (справочное) Типовые схемы выработки общего ключа с аутентификацией абонентов

Описываемый в настоящих рекомендациях 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 содержащейся в 5.6 последовательности действий, но реализовывать при этом только один из возможных принципов аутентификации. Ниже мы приводим типовые схемы реализации протокола выработки ключей.

А.1 Схема аутентификации на основе предварительно распределенного ключа

Схема с аутентификацией на основе предварительно распределенного ключа может использоваться в устройствах, использующих для взаимной аутентификации клиента и сервера предварительно распределенный секретный ключ PSK. 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.

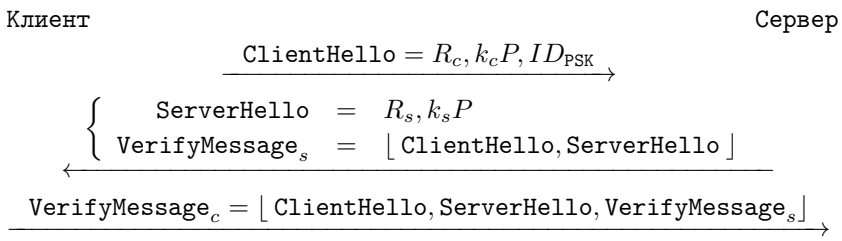

Рисунок А1. Схема аутентификации на основе предварительно распределенного ключа. Сеансовый уровень

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 5.1.1.

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 А2. 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. 5.4.

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.

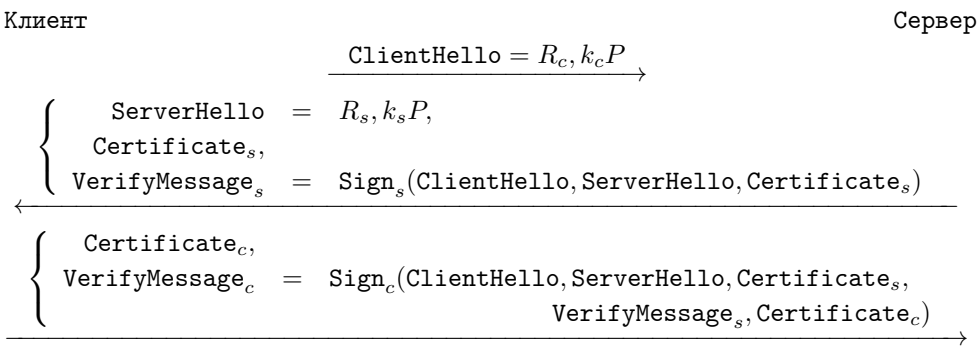

Рисунок А3. Схема аутентификации на основе ключа проверки электронной подписи. Сеансовый уровень

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.

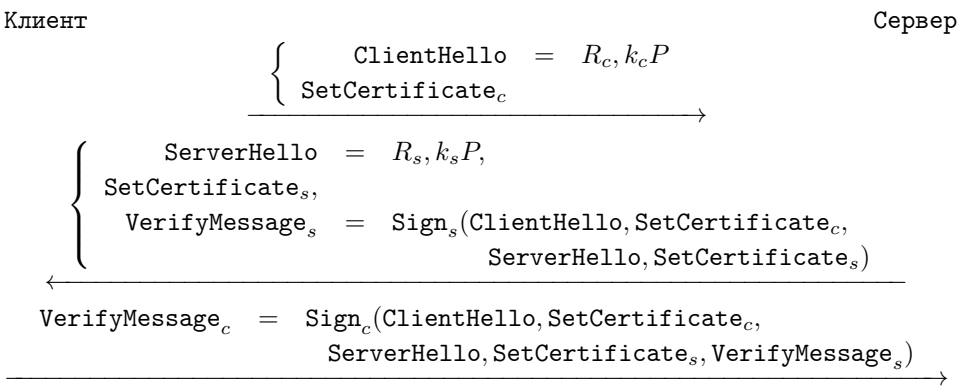

Рисунок А.5. Схема со взаимной аутентификацией на основе предварительно распределенных ключей проверки электронной подписи. Сеансовый уровень

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

Б Приложение (справочное) Механизмы формирования предварительно распределенных ключей

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

Настоящее приложение содержит один из допустимых механизмов формирования предварительно распределенных ключей аутентификации, являющийся модификацией подхода, регламентированного рекомендациями [13].

Мы предполагаем, что при взаимодействии контрольные и измерительные устройства выступают в роли клиентов и передают информацию единому центру, в качестве которого выступает сервер. Регламентированный в [13] подход позволяет выработать уникальные ключи для связи каждого контрольного или измерительного устройства с сервером, зависящие от уникальных идентификаторов участников взаимодействия.

Б.1 Идентификация участников защищенного взаимодействия

Для идентификации контрольных и измерительных устройств мы используем следующую двух уровневую схему:

а) в качестве идентификатора первого уровня должна выступать уникальная непустая последовательность октетов $ID_p \in \mathbb{B}^*$, содержащая, например, информацию о производителе устройств, номере серии устройств, дате ввода серии устройств в эксплуатацию и т.п; мы будем считать, что идентификатор первого уровня идентифицирует некоторое большое множество (серию) контрольных и измерительных устройств;

б) в качестве идентификатора второго уровня должна выступать уникальная непустая последовательность октетов $ID_u \in \mathbb{B}^*$, содержащая информацию о контрольном или измерительном устройстве, например, 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.

Тогда в качестве уникального идентификатором контрольного или измерительного устройства выступает последовательность октетов

$$ID_c = \mathtt{Streebog}_{256}(ID_p||ID_u),$$

являющаяся конкатенацией идентификаторов первого и второго уровней.

В качестве идентификатора сервера выступает произвольная непустая строка октетов $ID_s \in \mathbb{B}^*$, содержащая, например, информацию о производителе устройств, номер принимающего информацию устройства и т.п.

Б.2 Операции в конечном поле $\mathbb{F}_{2^{256}}$

Каждый двоичный вектор из $\mathbb{V}^{256}$ может быть представлен в виде элемента конечного поля $\mathbb{F}_{2^{256}}$. Данное представление взаимно однозначно и может быть задано следующим образом.

Пусть вектор $a = (a_0, \ldots, a_{255}) \in \mathbb{V}^{256}$, тогда ему будет соответствовать многочлен $a(x) = \sum_{i=0}^{255} a_i x^i \in \mathbb{F}_2[x]$, $\deg a(x) \leq 255$. Используя данное соответствие определим операции сложения и умножения двоичных векторов из $\mathbb{V}^{256}$ следующим образом.

Определим неприводимый многочлен:

$$p(x) = x^{256} + x^{10} + x^5 + x^2 + 1 \in \mathbb{F}_2[x].$$

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

$$a(x) = \sum_{i=0}^{255} a_i x^i, \quad b(x) = \sum_{i=0}^{255} b_i x^i, \quad a(x), b(x) \in \mathbb{F}_2[x],$$

тогда

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

  $$c(x) = \sum_{i=0}^{255} c_i x^i = a(x) + b(x) \pmod{p(x)},$$

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

- вектор ab определяется равенством $ab = c$, где $c = (c_0, \ldots, c_{255})$ есть вектор, которому соответствует многочлен

  $$c(x) = \sum_{i=0}^{255} c_i x^i = a(x)b(x) \pmod{p(x)},$$

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

Б.3 Ключевая система

Ключевая система, используемая при взаимодействии контрольных и измерительных устройств, представляет собой вариант схемы распределения ключей Блома, см. [14], и состоит из следующих секретных ключей, используемых для выработки уникальных ключей аутентификации.

а) Пусть u натуральное число. Мастер-ключ MK представляет собой симметричную матрицу A размера $(u+1) \times (u+1)$

$$A = \begin{pmatrix} a_{00}\ a_{01} \ldots\ a_{0u} \\ a_{10}\ a_{11} \ldots\ a_{1u} \\ \ldots \\ a_{u0}\ a_{u1} \ldots\ a_{uu} \end{pmatrix},$$

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

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 $\frac{1}{2}(u+1)(u+2)$.

2. Коэффициенты матрицы A однозначно связаны с многочленом:

$$f(x,y) = \sum_{i=0}^{u}\sum_{j=0}^{u} a_{ij}x^i y^j, \quad f(x,y) \in \mathbb{F}_{2^{256}}[x,y]. \tag{1}$$

3. Значение мастер-ключа 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 положениями раздела 5.3 рекомендаций [1].

б) Ключ сервера, предназначенный для выработки уникальных ключей аутентификации.

Каждому серверу соответствует свой собственный уникальный ключ K_s, зависящий от мастер-ключа MK и идентификатора сервера ID_s. Ключ K_s представляет собой вектор:

$$(c_0, \ldots, c_m), \quad c_j \in \mathbb{V}^{256}, \quad j = 0, \ldots, m,$$

определяемый равенством:

$$f(\mathtt{Streebog}_{256}(ID_s), y) = \sum_{j=0}^{m} c_j y^j \in \mathbb{F}_{2^{256}}[y],$$

где многочлен $f(x,y)$ определяется равенством (1).

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

$$c_j = \sum_{i=0}^{m} a_{ij}\big(\mathtt{Streebog}_{256}(ID_s)\big)^i,$$

выполненное для всех $j = 0, \ldots, m$.

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 MK в условиях, указанных в примечании 3 к перечислению а).

в) Ключ серии контрольных и измерительных устройств, предназначенный для выработки уникальных ключей аутентификации.

Каждой серии контрольных и измерительных устройств соответствует свой собственный уникальный ключ K_{cp}, зависящий от мастер-ключа MK и идентификатора первого уровня ID_p. Ключ серии K_{cp} представляет собой вектор:

$$(b_0, \dots, b_m), \quad b_i \in \mathbb{V}^{256}, \quad i = 0, \dots, m,$$

определяемый равенством:

$$f(x, \mathtt{Streebog}_{256}(ID_p)) = \sum_{i=0}^{m} b_i x^i \in \mathbb{F}_{2^{256}}[x],$$

где многочлен $f(x, y)$ определен равенством (1).

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

$$b_i = \sum_{j=0}^{m} a_{ij} \big(\mathtt{Streebog}_{256}(ID_p)\big)^j,$$

выполненное для всех $i = 0, \dots, m$.

Примечание. Значение ключа серии K_{cp} не должно быть известно участникам защищенного взаимодействия. Формирование ключа серии K_{cp} должно производиться непосредственно после выработки мастер-ключа MK в условиях, указанных в примечании 3 к перечислению а).

г) Ключ аутентификации $ePSK$.

Ключ аутентификации $ePSK \in \mathbb{V}256$ контрольного или измерительного устройства представляет собой двоичную последовательность длины 256 бит и вырабатывается с помощью следующих значений:

- идентификатора ID_p первого уровня, содержащего информацию о серии устройств, которой принадлежит данное устройство;
- идентификатора ID_u второго уровня, содержащего уникальную информацию об устройстве;
- идентификатора сервера ID_s;
- ключа серии $K_s = (b_0, \dots, b_m)$, $b_i \in \mathbb{V}^{256}$, $i = 0, \dots, m$;

Ключ аутентификации $\mathtt{ePSK}$ определяется равенством

$$\mathtt{ePSK} = \mathtt{HMAC}_{256}(K^*, \mathtt{0x01}||ID_p||\mathtt{0x00}||ID_u||\mathtt{0x00}||\mathtt{0x01}),$$

где:

$$K^* = \sum_{i=0}^{m} b_i \left(\mathtt{Streebog}_{256}(ID_s)\right)^i \in \mathbb{V}^{256}.$$

Идентификатором ключа $ID_{\mathtt{ePSK}}$ является последовательность октетов, определяемая соотношением

$$ID_{\mathtt{ePSK}} = ID_p||ID_u.$$

Примечания

1. Ключ аутентификации ePSK существенным образом зависит от идентификаторов ID_p, ID_u, ID_s. Такая зависимость позволяет однозначно связать значение ключа аутентификации с фиксированными участниками защищенного взаимодействия.

2. Преобразование, определяющее значение ключа аутентификации ePSK является частным случаем алгоритма диверсификации KDF_TREE_GOST3411_2012_256, определяемого рекомендациями [6].

3. Эквивалентным определением величины $K^* \in \mathbb{V}^{256}$ является равенство:

$$K^* = f(\texttt{Streebog}_{256}(ID_s), \texttt{Streebog}_{256}(ID_p)) \in \mathbb{V}^{256}.$$

4. Ключ аутентификации ePSK должен быть известен только серверу и клиенту, с которым выполняется защищенное взаимодействие. Формирование ключ аутентификации ePSK должно производиться непосредственно после выработки мастер-ключа MK и ключа серии K_{cp} в условиях, указанных в примечании 3 к перечислению а). Ключ аутентификации ePSK должен записываться непосредственно в блок СКЗИ контрольного или измерительного устройства.

5. Для идентификатора $ID_{\texttt{ePSK}}$ ключа аутентификации ePSK и идентификатора клиента ID_c выполнено равенство

$$ID_c = \texttt{Streebog}_{256}(ID_{\texttt{ePSK}}).$$

Данное соотношение позволяет серверу вычислять ключ аутентификации ePSK по его идентификатору в ходе выполнения протокола выработки ключей, а не хранить в памяти заранее вычисленное значение.

В Приложение (обязательное) Форматы передаваемых данных

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 С, см. [3], нотации. Указанные структуры данных предполагаются упакованными, то есть занимающими в памяти вычислительного средства объем, являющийся в точности суммой объемов памяти, занимаемых элементами структур данных, см. [3].

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.1 Базовые типы данных

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

В.1.1 Описатель `Octet`

```
typedef unsigned char Octet;
```

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

В.1.2 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, в 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.1.3 Описатель `LengthOctet`

```
typedef Octet LengthOctet;
```

Описатель `LengthOctet` предназначен для определения целых неотрицательных чисел, как правило длин последовательностей, не превосходящих 255. Октет данного типа интерпретируется как целое неотрицательное число, принадлежащее интервалу от 0 до 255.

B.1.4 Описатель `LengthShortInt`

```
typedef Octet LengthShortInt[2];
```

Описатель `LengthShortInt` предназначен для определения последовательностей, состоящую из двух октетов. Данная последовательность предназначена 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

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

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

$$\texttt{len}[0] = \frac{l - (l \pmod{256})}{256}, \quad \texttt{len}[1] = l \pmod{256}.$$

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

B.1.5 Описатель `RandomOctetString`

```
typedef Octet RandomOctetString[32];
```

Описатель `RandomOctetString` определяет последовательность октетов фиксированной длины (32 октета). Данная последовательность используется для хранения и передачи по каналам связи данных, выработанных с использованием генераторов случайных чисел.

B.2 Перечислимые и служебные типы

B.2.1 Описатель `FrameType`

```
typedef enum {
  plainFrame = 0xA0,
  encryptedFrame = 0xA2
} FrameType;
```

Описатель `FrameType` предназначен для определения типа передаваемого фрейма данных, см. тип данных `Frame`, раздел B.3.1. 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.

B.2.2 Описатель `PresentType`

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

Описатель `PresentType` предназначен для указания установлено ли значение некоторой опциональной переменной или же 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.

B.2.3 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 производится;

б) `isRequested` – запрос идентификатора выполняется.

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

B.2.4 Описатель `CertificateFormat`

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

Описатель `CertificateFormat` предназначен указания формата используемого сертификата ключа проверки электронной подписи, см. раздел 5.1.1. Указанные выше константы описывают следующие возможности:

а) `plain` – сертификатом ключа проверки электронной подписи является последовательность октетов, полученная в результате сериализации объекта типа `EllipticCurvePoint`, см. раздел В.2.10;

б) `x509` – сертификат ключа проверки электронной подписи соответствует проекту рекомендаций [4];

в) `cvc` – сертификат ключа проверки электронной подписи соответствует формату сертификатов, рекомендуемых к применению в контрольных и измерительных устройствах, см. [5].

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

В.2.5 Описатель `CertificateProcessedType`

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

Описатель `CertificateProcessedType` предназначен 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.2.6 Описатель `Certificate`

```
typedef OctetString Certificate;
```

Описатель `Certificate` определяет последовательность октетов произвольной конечной длины, предназначенную для представления сертификата ключа проверки электронной подписи, см. раздел 5.1.1. Допускается использование сертификатов

ключей проверки электронной подписи, хранящихся в различных форматах. Перечень допустимых форматов определяется в разделе B.2.4.

Примечание. Сертификат ключа проверки электронной подписи передается от одного абонента к другому при помощи расширения `CertificateExtension`, см. разделы 5.6.2 и 5.6.3. Поскольку длина расширения `CertificateExtension` указывается при его вложении во фрейм, см. раздел B.3.1, это позволяет однозначно восстановить длину передаваемого сертификата ключа проверки электронной подписи.

B.2.7 Описатель `MessageType`

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

Описатель `MessageType` предназначен 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. раздел B.4.1, определяющая формат сообщения, с помощью которого клиент инициализирует выполнение защищенного взаимодействия;

б) `serverHello` – структура `ServerHelloMessage`, см. раздел B.4.2, определяющая формат сообщения, с помощью которого сервер передает ответ на инициализирующий запрос клиента;

в) `verify` – структура `VerifyMessage`, см. раздел B.4.3, определяющая формат сообщения, содержащего один или несколько кодов аутентификации;

г) `applicationData` – структура `ApplicationDataMessage`, см. раздел B.4.4, определяющая формат сообщений, передаваемых в ходе выполнения протокола передачи прикладных данных.

д) `alert` – структура `AlertMessage`, см. раздел B.4.5, определяющая формат сообщения, содержащего код ошибки выполнения протокола;

е) `generatePSK` – структура `GeneratePSKMessage`, см. раздел В.4.6, определяющая формат сообщения, передаваемого в процессе протокола выработки ключа аутентификации;

ж) `extensionRequestCertificate` – структура `RequestCertificateExtension`, см. раздел В.5.1, определяющая формат расширения, используемого для запроса сертификата ключа проверки электронной подписи;

з) `extensionCertificate` – структура `CertificateExtension`, см. раздел В.5.2, определяющая формат расширения, используемого для передачи сертификатов ключей проверки электронной подписи;

и) `extensionSetCertificate` – структура `SetCertificateExtension`, см. раздел В.5.3, определяющая формат расширения, используемого для указания использования конкретного сертификата ключа проверки электронной подписи;

к) `extensionInformCertificate` – структура `InformCertificateExtension`, см. раздел В.5.4, определяющая формат расширения, используемого для информирования о номере используемого сертификата ключа проверки электронной подписи;

л) `extensionRequestIdentifier` – структура `RequestIdentifierExtension`, см. раздел В.5.5, определяющая формат расширения, используемого для запроса и/или указания используемого идентификатора абонента;

м) `extensionKeyMechanism` – структура `KeyMechanismExtension`, см. раздел В.5.6, 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.2.8 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. тип данных `LengthShortInt`, 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 9.

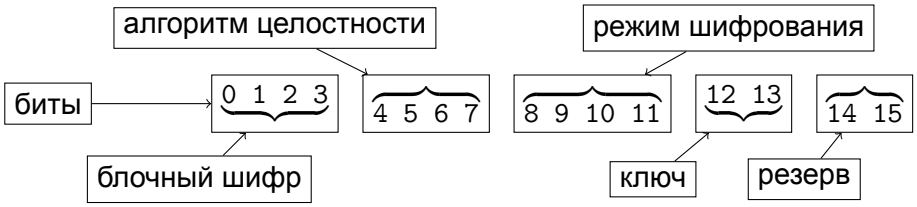

Рисунок 9. Порядок назначения значений бит в определении криптографического механизма.

1. Для используемого алгоритма блочного шифрования допускаются следующие значения параметров:

а) `0x0` – алгоритм блочного шифрования не определен;

б) `0x1` – используется алгоритм блочного шифрования «Магма», регламентируемый стандартом ГОСТ Р 34.12 – 2015;

в) `0x2` – используется алгоритм блочного шифрования «Кузнечик», регламентируемый стандартом 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 стандартом ГОСТ Р 34.11 – 2012;

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

г) `0x3` – используется ключевая функция хеширования $\mathtt{HMAC}_{256}$ с длиной кода 256 бит, регламентируемая рекомендациями [6, раздел 4.1.1];

д) 0x4 – используется ключевая функция хеширования HMAC_{512} с длиной кода 512 бит, регламентируемая рекомендациями [6, раздел 4.1.2];

е) 0x5 – используется ключевая функция хеширования, регламентируемая стандартом ГОСТ Р 34.13 – 2015, раздел 5.6.

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 стандартом ГОСТ Р 34.13 – 2015, раздел 5.2.

в) 0x2 – режим шифрования с возможностью одновременной выработки имитовставки, реализуемый в соответствии с [7].

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. раздел 6.4;

в) 0x2 – используется ключ аутентификации ePSK, см. раздел 5.1.1;

г) 0x3 – используется ключ аутентификации iPSK, см. раздел 5.1.1.

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

B.2.9 Описатель EllipticCurveID

```
typedef enum {
  tc26_gost3410_2012_256_paramsetA = 0x01,
  tc26_gost3410_2012_512_paramsetA = 0x02,
  tc26_gost3410_2012_512_paramsetB = 0x03,
  tc26_gost3410_2012_512_paramsetC = 0x04,
  rfc4357_gost3410_2001_paramsetA = 0x05,
  rfc4357_gost3410_2001_paramsetB = 0x06,
  rfc4357_gost3410_2001_paramsetC = 0x07
} EllipticCurveID;
```

Описатель EllipticCurveID предназначен для указания используемых в протоколе выработки общих ключей, см. раздел 5, параметров эллиптической кривой. Под параметрами эллиптической кривой подразумевается следующий набор значений:

а) простое число p;

б) коэффициенты эллиптической кривой:

- либо пара $a, b \in \mathbb{F}_p$, определяющая эллиптическую кривую E, заданную в канонической форме Вейерштрасса сравнением

$$y^2 \equiv x^3 + ax + b \pmod{p},$$

- либо пара $e, d \in \mathbb{F}_p$, определяющая эллиптическую кривую E, заданную в форме скрученной кривой Эдвардса сравнением

$$eu^2 + v^2 \equiv 1 + du^2v^2 \pmod{p};$$

в) m – порядок группы точек эллиптической кривой E;

г) точка эллиптической кривой P, заданная парой своих координат:

- либо $x, y \in \mathbb{F}_p$ — для эллиптической кривой, заданной в канонической форме Вейерштрасса,
- либо $u, v \in \mathbb{F}_p$ — для эллиптической кривой, заданной в форме скрученной кривой Эдвардса;

д) простое число q, определяющее порядок группы, порожденной точкой P.

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

- методическими рекомендациями [2, раздел 3.1] — эллиптические кривые, заданные в канонической форме Вейерштрасса
  - id-tc26-gost-3410-2012-512-paramSetA,
  - id-tc26-gost-3410-2012-512-paramSetB;
- методическими рекомендациями [2, раздел 4.1] — эллиптические кривые, заданные в форме скрученной кривой Эдвардса
  - id-tc26-gost-3410-2012-256-paramSetA,
  - id-tc26-gost-3410-2012-512-paramSetC;
- рекомендациями [8, раздел 11.4] — эллиптические кривые, заданные в канонической форме Вейерштрасса
  - id-rfc4357-gost3410-2001-paramsetA,
  - id-rfc4357-gost3410-2001-paramsetB,
  - id-rfc4357-gost3410-2001-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.

B.2.10 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

$$\texttt{point} = \texttt{id}||\texttt{x}||\texttt{y},$$

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), и

При этом мы предполагаем, что переменные `x` и `y` типа `OctetString` представляют собой результат сериализации определяющих точку эллиптической кривой вычетов x и y, которые удовлетворяют равенствам

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

B.2.11 Описатель PreSharedKeyID

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

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

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

б) length – длина идентификатора (количество октетов, занимаемых полем id);

в) id – произвольная последовательность октетов конечной длины, определяемой значением length.

Примечание. Если в качестве предварительно распределенного ключа выступает ключ iPSK (предварительно распределенный ключ, выработанный в ходе выполнения предыдущего сеанса защищенного взаимодействия), то его идентифкатором является последовательность октетов длины 32, см. раздел 6.5.

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

$$\texttt{keyid} = \texttt{present},$$

если $\texttt{present} = \texttt{notPresent}$, либо

$$\texttt{keyid} = \texttt{present}||\texttt{length}||\texttt{id},$$

если $\texttt{present} = \texttt{isPresent}$ и

```
          keyid[0] = present;
          keyid[1] = length;
          keyid[2] = id[0];
                 .....
keyid[length + 2] = id[length - 1];
```

B.2.12 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 (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 length. Точное значение длины кода целостности (имитовставки) определяется выбранным клиентом или сервером криптографическим алгоритмом, см. тип данных CryptoMechanism.

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

$$\texttt{icode} = \texttt{present},$$

если present = notPresent, либо

$$\texttt{keyid} = \texttt{present}||\texttt{length}||\texttt{code},$$

если present = isPresent и

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

B.2.13 Описатель FrameNumber

```
typedef Octet FrameNumber[5];
```

Описатель FrameNumber определяет последовательность из пяти октетов, предназначенную для указания криптографических номеров фреймов, см. раздел B.3.1. 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 разделе 7.2.

B.2.14 Описатель `KeyMechanismType`

```
typedef enum {
  standard221 = 0x01,
  shortKCmagma = 0x02,
  shortKCkuznechik = 0x03,
  longKCmagma = 0x04,
  longKCkuznechik = 0x05,
  shortKAmagma = 0x06,
  shortKAkuznechik = 0x07,
  longKAmagma = 0x08,
  longKAkuznechik = 0x09
} 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.2.15 Описатель `AlertType`

```
typedef enum {
  unknownError = 0x1000,
  unsupportedCryptoMechanism = 0x1001,
  wrongExternalPreSharedKey = 0x1002,
  wrongInternalPreSharedKey = 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,
  wrongInternalPSKIdentifier = 0x1040
} 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 длины два 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 по стандартизации;
- usupportedCryptoMechanism – использование недопустимого криптографического механизма;
- wrongExternalPreSharedKey – использование неверного идентификатора предварительно распределенного ключа аутентификации;
- wrongInternalPreSharedKey – использование неверного идентификатора ключа аутентификации, выработанного в ходе предыдущего сеанса защищенного взаимодействия;
- 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 – не поддерживаемый номер сертификата ключа проверки электронной подписи;
- notValidCertificateNumber – недопустимый номер сертификата ключа проверки электронной подписи;
- wrongCertificateApplication – неверная область использования 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 не поддерживаемым центром сертификации (идентификатором удостоверяющего центра);
- `unsupportedCertificateFormat` – запрос или указание на использование сертификата с неизвестным или не поддерживаемым форматом;
- `wrongCertificateIntegrityCode` – запрос или указание на использование сертификата, содержащего неверное значение электронной подписи;
- `usupportedKeyMechanism` – запрос на использование неизвестного или не поддерживаемого набора параметров криптографических механизмов;
- `unsupportedEllipticCurveID` – переданные данные содержат неверный или не поддерживаемый идентификатор эллиптической кривой;
- `wrongEllipticCurvePoint` – переданные данные содержат неверную точку эллиптической кривой;
- `wrongInternalPSKIdentifier` – переданный идентификатор ключа аутентификации не может быть подтвержден.

B.3 Формат сообщений транспортного протокола

B.3.1 Фрейм – `Frame`

```
typedef struct {
  FrameType      tag;
  LengthShortInt length;
  FrameNumber    number;
  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 B.4.

Процедура вложения сообщения во фрейм описывается в 7.3.1. Процедура получения сообщения из фрейма описывается в 7.4.

Структура данных `Frame` состоит из следующих полей:

а) `tag` — тип фрейма, см. B.2.1, представленный в виде одного октета с фиксированным значением;

б) `length` — длина всего фрейма (в октетах), представленная в виде целого числа, записываемого двумя октетами, см. B.1.4;

в) `number` — криптографический номер фрейма, см. B.2.13, который позволяет однозначно связать данный фрейм с набором криптографических ключей, используемых для обеспечения конфиденциальности и целостности передаваемой информации;

г) `type` — тип вложенного во фрейм сообщения, см. B.2.7;

д) `meslen` — длина вложенного во фрейм сообщения (в октетах), представленная в виде целого числа, записываемого двумя октетами, см. B.1.4;

е) `message` — вложенное во фрейм сообщение;

ж) `padding` — произвольная последовательность октетов конечной длины (дополнение);

з) `icode` — код целостности или имитовставка, см. B.2.12;

При сериализации объект типа `Frame` преобразуется в последовательность октетов путем последовательной конкатенации своих полей, см. рисунок 10.

$$\texttt{frame} = \underbrace{\texttt{tag}||\texttt{length}||\texttt{number}}_{\text{заголовок}} || \underbrace{\texttt{type}||\texttt{meslen}||\texttt{message}||\texttt{padding}}_{\text{тело}} ||\texttt{icode}$$

Рисунок 10. Формат фрейма.

Для описания взаимосвязи между длинами различных фрагментов фрейма определим следующие величины:

- $l = \texttt{Len}(\texttt{frame})$ — длина всего фрейма,
- $m = \texttt{Len}(\texttt{meslen})$ — длина сообщения, вкладываемого во фрейм;
- $p = \texttt{Len}(\texttt{padding})$ — длина дополнения;
- $i = \texttt{Len}(\texttt{icode})$ — длина сериализованного представления типа `IntegrityCode`, содержащего значение кода целостности,

тогда выполнено равенство $l = 11+m+p+i$. Данное равенство позволяет однозначно восстановить длину дополнения, при получении фрейма, поскольку $p = l-(11+m+i)$.

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] = type;
```

```
      frame[9] = meslen[0];
     frame[10] = meslen[1];
     frame[11] = message[0];
               .....
   frame[m+10] = message[m-1];
   frame[m+11] = padding[0];
               .....
frame[m+p+10] = padding[p-1];
frame[m+p+11] = icode[0];
               .....
    frame[l-1] = icode[i-1]; /* l = 11 + m + p + i */
```

Примечание. При передаче по каналу связи первые восемь октетов фрейма, с нулевого октета по седьмой, всегда передаются в незашифрованном виде. Эта часть фрейма называется его заголовком.

Октеты, содержащие тип и длину сообщения, само сообщение и дополнение к нему могут передаваться как в зашифрованном, так и в открытом виде. Эта часть фрейма называется его телом. Последний фрагмент фрейма называется кодом целостности фрейма.

B.4 Формат сообщений протоколов сеансового уровня

B.4.1 Сообщение `ClientHelloMessage`

```
typedef struct {
  CryptoMechanism    algorithm;
  PreSharedKeyID     idipsk;
  PreSharedKeyID     idepsk;
  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 разделе 5.6.1.

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) `random` – случайная (псевдослучайная) последовательность октетов фиксированной длины;

в) `point` – точка эллиптической кривой, используемая в протоколе Диффи-Хеллмана для выработки общего ключа. Данная точка определятся равенством:

$$P_c = k_c P,$$

где k_c случайное число, выработанное клиентом, а P – фиксированная точка эллиптической кривой, см. раздел 5.6.1;

г) `idipsk` – идентификатор ключа аутентификации `iPSK`, выработанного в ходе предыдущей сессии криптографического взаимодействия. Данный параметр является опциональным – если идентификатор не определен, то должно быть установлено значение `idipsk.present` = `notPresent`.

д) `idepsk` – идентификатор предварительно распределенного ключа аутентификации `ePSK`, выработка которого производилась независимо от выполнения протокола выработки ключей. Данный параметр является опциональным – если идентификатор не определен, то должно быть установлено значение `idepsk.present` = `notPresent`.

Примечание. Ключи аутентификации `iPSK` и `ePSK` определяются в 5.1.1. Механизмы формирования ключей `iPSK` и `ePSK` описываются, соответственно, в 6.5 и приложении Б.

е) `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 – переменная типа `ClientHelloMessage` преобразуется в последовательность октетов следующим образом.

$$\texttt{clienthello} = \texttt{algorithm}||\texttt{idipsk}||\texttt{idepsk}||\texttt{random}||\texttt{point}||\texttt{countOfExtension},$$

где `p = Len(point)`, `i = Len(idipsk)`, `e = Len(idepsk)` – длины соответствующих полей сообщения `ClientHelloMessage`.

```
clienthello[0] = algorithm[0];
clienthello[1] = algorithm[1];
clienthello[2] = iPSK[0];
                 .....
clienthello[1 + i] = iPSK[i-1];
clienthello[2 + i] = ePSK[0];
                 .....
clienthello[1 + i + e] = ePSK[e-1];
clienthello[2 + i + e] = random[0];
                 .....
clienthello[33 + i + e] = random[31];
clienthello[34 + i + e] = point[0];
                 .....
```

```
clienthello[33 + i + e + p] = point[p-1];
clienthello[34 + i + e + p] = countOfExtension;
```

Кроме того, выполнено равенство `Len(clienthello) = 35 + p + i + e`.

B.4.2 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 разделе 5.6.2.

Структура данных `ServerHelloMessage` состоит из следующих полей:

а) `algorithm` – алгоритм шифрования с возможностью одновременной выработки имитовставки, используемый для шифрования и подтверждения целостности данных. Возможные значения данного поля определяются типом `CryptoMechanism`;

б) `random` – случайная (псевдослучайная) последовательность октетов фиксированной длины;

в) `point` – точка эллиптической кривой, используемая в протоколе Диффи-Хеллмана для выработки общего ключа. Данная точка определяется равенством:

$$P_s = k_s P,$$

где k_s случайное число, выработанное сервером, а P – фиксированная точка эллиптической кривой, см. раздел 5.6.2;

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.

$$\texttt{serverhello} = \texttt{algorithm}||\texttt{random}||\texttt{point}||\texttt{countOfExtension},$$

где `p = Len(point)` – длина соответствующего поля сообщения `ServerHelloMessage`.

```
serverhello[0] = algorithm[0];
serverhello[1] = algorithm[2];
serverhello[2] = random[0];
                .....
serverhello[33] = random[31];
serverhello[34] = point[0];
                .....
serverhello[p + 33] = point[p - 1];
serverhello[p + 34] = countOfExtension;
```

Кроме того, выполнено равенство `Len(serverhello) = 34 + p`.

B.4.3 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 5.6.2 или 5.6.3; данное значения является опциональным;

б) `sign` – определяемое типом `IntegrityCode` значение электронной подписи, выработанное в соответствии с 5.6.2 или 5.6.3; данное значения 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.

$$\texttt{verify} = \texttt{mac} || \texttt{sign},$$

где `m = Len(mac)`, `s = Len(sign)` – длины соответствующих полей сообщения `VerifyMessage` и

```
verify[0] = mac[0];
            .....
```

```
      verify[m-1] = mac[m-1];
        verify[m] = sign[0];
                  .....
verify[m + s - 1] = sign[s - 1];
```

Кроме того, выполнено равенство `Len(verify) = m + s`.

B.4.4 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` и передаваться по каналам связи только в зашифрованном виде.

B.4.5 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. B.2.8, используемый для контроля целостности сообщений, передаваемых в открытом виде; данное значение может отличаться от значения, выбранного клиентом при инициализации протокола выработки общих ключей, см. 5.6.1;

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, то есть

```
alert = code||algorithm||present||message
```

и имеет вид

```
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 $l = \mathrm{Len}(\mathtt{message})$.

B.4.6 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, см. 6.5.

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; используется при выработке значения ключа аутентификации iPSK;

б) id – идентификатор выработанного ключа аутентификации iPSK; возвращается инициатору протокола для подтверждения корректности выработки ключа аутентификации.

При сериализации типа данных GeneratePSKMessage используется последовательная конкатенация полей структуры – переменная типа GeneratePSKMessage преобразуется в последовательность октетов следующим образом:

$$\mathtt{pskmessage} = \mathtt{random} || \mathtt{id}$$

и

```
pskmessage[0] = random[0];
              .....
pskmessage[31] = random[31];
pskmessage[32] = id[0];
              .....
pskmessage[l + 31] = id[l - 1],
```

где l – длина последовательности октетов id.

B.5 Формат расширений

B.5.1 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. 5.3.1.

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:

$$\texttt{reqcertext} = \texttt{certproctype}||\texttt{identifier},$$

и

```
reqcertext[0] = certproctype;
reqcertext[1] = identifier[0];
              .....
reqcertext[l] = idetifier[l - 1];
```

где $l = \texttt{Len}(\texttt{identifier})$.

B.5.2 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 5.3.1.

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.

$$\texttt{certext} = \texttt{format}||\texttt{certificate},$$

и, обозначая $l = \texttt{Len}(\texttt{certificate})$,

```
certext[0] = format;
certext[1] = certificate[0];
           .....
certext[l] = certificate[l-1];
```

Кроме того, выполнено равенство $\texttt{Len}(\texttt{certext}) = l + 1$.

B.5.3 Расширение SetCertificateExtension

```
typedef RequestCertificateExtension SetCertificateExtension;
```

или

```
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. 5.3.1.

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:

$$\texttt{setcertext} = \texttt{certproctype}||\texttt{identifier},$$

и

```
setcertext[0] = certproctype;
setcertext[1] = identifier[0];
              .....
setcertext[l] = idetifier[l - 1];
```

где $l = \texttt{Len}(\texttt{identifier})$.

B.5.4 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. 5.3.1.

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.5.5 Расширение `RequestIdentifierExtension`

```
typedef strust{
  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. 5.4.

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.

$$\texttt{reqidext} = \texttt{request}||\texttt{identifier},$$

и

```
reqidext[0] = request;
reqidext[1] = identifier[0];
              .....
reqidext[l] = idetifier[l - 1];
```

где $\mathtt{l} = \mathtt{Len}(\mathtt{identifier})$.

B.5.6 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 5.7.6.

При сериализации переменная типа данных `KeyMechanismExtension` представляется в виде одного октета, значение которого совпадает с константой из множества `KeyMechanismType`.

Г Приложение (справочное) Рекомендуемые значения параметров защищенного взаимодействия

В настоящем приложении мы приведем 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 [15], 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 по стандартизации [1] вводится классификация СКЗИ. На СКЗИ, не попадающие под действие Положения ПКЗ-2005, регулирование и классификация не распространяются. Далее мы приведем значения параметров защищенного взаимодействия для СКЗИ как попадающих, так и не попадающих под действие Положения ПКЗ – 2005.

Мы определяем значения следующих параметров:

- `maxFrameCount` — определяет максимально допустимое количество фреймов, конфиденциальность и целостность которых обеспечивается одной парой производных ключей шифрования и выработки имитовставки, см. 7.1;
- `maxFrameKeysCount` — определяет максимально допустимое количество пар производных ключей шифрования $\texttt{eSFK}_{n,m}$, $\texttt{eCFK}_{n,m}$ и производных ключей выработки имитовставки $\texttt{iSFK}_{n,m}$, $\texttt{iCFK}_{n,m}$ для одного фиксированного состояния ключевой информации $\texttt{CATS}_n$ и $\texttt{SATS}_n$, см. 6.2;
- `maxApplicationSecretCount` — определяет максимально возможное количество преобразований ключевой информации $\texttt{CATS}_n$ и $\texttt{SATS}_n$, допустимое в рамках одного сеанса защищенного взаимодействия, см. 6.2.

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. 7.1.

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 значением `standard221` перечисления `KeyMechanismType`. Это значение может использоваться определяемом в 5.7.6 расширении `KeyMechanismExtension`.

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».

Д Приложение (справочное) Контрольные примеры

В настоящем приложении приводятся контрольные примеры, иллюстрирующие процесс защищенного взаимодействия, в соответствии с приведенными в приложении А типовыми схемами выработки общего ключа.

Все данные приводятся в виде последовательностей октетов записанных слева направо – вначале записываются октеты с наименьшими адресами, потом с наибольшими адресами.

Короткие последовательности октетов, имеющие в своей записи не более двух октетов и начинающиеся с символа `0x`, имеют обратный порядок записи. Данное обозначение используется для указания целочисленных констант, определенных в приложении В.

Д.1 Пример взаимодействия с аутентификацией на основе предварительно распределенного ключа

В настоящем примере абоненты имеют следующие идентификаторы:

- сервер $ID_s =$ `7365727665722D303031`;
- клиент $ID_c =$ `636C69656E742D3030313337`.

Предварительно распределенный ключ аутентификации[6]

- принимает значение `ePSK` =

  `BB769493AF2499C3223648303DB2EFDD568B77843BE77731F4E1539961F8190C`,

- и имеет идентификатор `idepsk` = `3132372E302E302E31`.

В настоящем примере используется процедура генерации дополнения, см. 7.3.3, при которой длины всех фреймов выравниваются до значения, кратного 160 октетам.

Также, в настоящем примере используется способ формирования уникальных номеров фреймов, определяемый константой `standard221`, см. приложение Г.

Д.1.1 Формирование сообщения `ClientHelloMessage`

В соответствии с 5.6.1 клиент выполняет следующую последовательность шагов.

а) Клиент вырабатывает случайную последовательность октетов длины 32 октета

`AF56198E2AA4124597748DB8382989603D44D7B78224D0F154C622C32C85831E`.

б) Клиент выбирает идентификатор эллиптической кривой

[6]Идентификатор `idepsk` ключа аутентификации `ePSK` представляет собой последовательность ascii-кодов, образующих символьную строку «127.0.0.1». Значение ключа есть результат применения бесключевой функции хеширования «Стрибог-256» к значению идентификатора `idepsk`.

ID_E = id_rfc4357_gost3410_2001_paramsetA = 05.

в) Клиент вырабатывает случайное целое число k_c

71DA7AF3391C1CEF063FCB3F8C96F74C6894E07EE98551FCC9596468F35843A6.

г) Клиент вычисляет точку кривой $P_c = k_c P$ с координатами

x = DE277EB89968BBC60B3854283F855B028B2BDD781A9C3839FC41AD8B8EA32AF2,
y = 43CA69DCF666C981AD1D7861639A22B20358F4209A588D2CC94FA2F464FA1ACB.

д) Клиент определяет используемый для контроля целостности сообщений криптографический механизм hmac256ePSK = 0x2030 (в сетевом порядке следования октетов сериализованное представление числа 0x2030 имеет вид 3020).

е) Клиент формирует сообщение ClientHelloMessage, образованное конкатенацией следующих фрагментов

```
3020
B0
B1
09
3132372E302E302E31
AF56198E2AA4124597748DB8382989603D44D7B78224D0F154C622C32C85831E
05
DE277EB89968BBC60B3854283F855B028B2BDD781A9C3839FC41AD8B8EA32AF2
43CA69DCF666C981AD1D7861639A22B20358F4209A588D2CC94FA2F464FA1ACB
00
```

ж) Клиент передает сформированное сообщение ClientHelloMessage и ключ аутентификации ePSK на транспортный уровень для формирования фрейма.

Д.1.2 Формирование фрейма для сообщения ClientHelloMessage

а) Для формирования фрейма клиент использует следующие значения:

а.1) tag = 0xA0 — фрейм передается в незашифрованном виде;

а.2) length = 160 — общая длина фрейма составляет 160 октетов (в сетевом порядке следования октетов сериализованное представление числа 160 имеет вид 00A0);

а.3) значения счетчиков $l = 0$, $m = 0$, $n = 0$, используемых для формирования уникального номера фрейма;

а.4) type = clientHello = 0x11;

а.5) meslen = 112 — длина сообщения ClientHelloMessage в октетах (в сетевом порядке следования октетов сериализованное представление числа 112 имеет вид 0070).

б) Поскольку результат применения функции $\texttt{HMAC}_{256}$ занимает 32 октета, то сериализованное представление типа `IntegrityCode`, см. B.2.12, должно занимать 34 октета (два октета `B120` за которыми следует значение имитовставки).

Учитывая это, клиент вычисляет длину дополнения p, см. B.3.1, равную

$$p = 160 - \texttt{meslen} - 11 - 34 = 3.$$

в) Клиент вырабатывает 3 случайных октета дополнения

$$\texttt{padding} = 246455$$

г) Клиент формирует заголовок и тело фрейма, образованные конкатенацией следующих фрагментов

```
A0
00A0
0000000000
11
0070
3020B0B1093132372E302E302E31AF56198E2AA4
124597748DB8382989603D44D7B78224D0F154C6
22C32C85831E05DE277EB89968BBC60B3854283F
855B028B2BDD781A9C3839FC41AD8B8EA32AF243
CA69DCF666C981AD1D7861639A22B20358F4209A
588D2CC94FA2F464FA1ACB00
246455
```

д) Для указанной последовательности октетов клиент вычисляет значение имитовставки, являющееся результатом применения алгоритма $\texttt{HMAC}_{256}$ на ключе `ePSK`

$$\texttt{62FD7DC1BF74221682393EC5DF66605B4111212647E7B5EE092E7A23F759E0C4}.$$

е) Клиент окончательно формирует тело фрейма, образованное конкатенацией следующих фрагментов

```
A000A0000000000001100703020B0B1093132372E
302E302E31AF56198E2AA4124597748DB8382989
603D44D7B78224D0F154C622C32C85831E05DE27
7EB89968BBC60B3854283F855B028B2BDD781A9C
3839FC41AD8B8EA32AF243CA69DCF666C981AD1D
7861639A22B20358F4209A588D2CC94FA2F464FA
1ACB00246455B12062FD7DC1BF74221682393EC5
DF66605B4111212647E7B5EE092E7A23F759E0C4
```

Д.1.3 Формирование сообщения ServerHelloMessage

В соответствии с 5.6.2 сервер проверяет корректность полученного сообщения, после чего выполняет следующую последовательность шагов.

а) Сервер вырабатывает случайную последовательность октетов длины 32 октета

95DEC4E0AF189B94D9EDC0FA915C2FEAC20232B686D922F0E5FC25299360F0AF.

б) Сервер вырабатывает случайное целое число k_s

B49B854C9A9FF50D837E1DF75F266BD1862598E085656EA5D1EF83A090ABFA0E.

в) Сервер вычисляет точку кривой $P_s = k_s P$ с координатами

x = 6C90280921184E36FFAB39F9728346388C87659C0209C2B9289D6A4D277596D2,
y = 6130E33AE93B4E2CE8D6BD8B2F8E3A3C0ECB203AB839A3A3F09F9E489FD97304.

г) Сервер определяет криптографические механизмы, которые будут использованы для передачи зашифрованных сообщений kuznechikCTRplusGOST3413 = 0x1152 (в сетевом порядке следования октетов сериализованное представление числа 0x1152 имеет вид 1152).

д) Сервер формирует сообщение ServerHelloMessage, образованное конкатенацией следующих фрагментов

```
5211
95DEC4E0AF189B94D9EDC0FA915C2FEAC20232B686D922F0E5FC25299360F0AF
05
6C90280921184E36FFAB39F9728346388C87659C0209C2B9289D6A4D277596D2
6130E33AE93B4E2CE8D6BD8B2F8E3A3C0ECB203AB839A3A3F09F9E489FD97304
00
```

е) Сервер передает сформированное сообщение ServerHelloMessage и ключ аутентификации ePSK на транспортный уровень для формирования фрейма.

Д.1.4 Формирование фрейма для сообщения ServerHelloMessage

а) Для формирования фрейма сервер использует следующие значения:

а.1) tag = 0xA0 — фрейм передается в незашифрованном виде;

а.2) length = 160 — общая длина фрейма составляет 160 октетов (в сетевом порядке следования октетов сериализованное представление числа 160 имеет вид 00A0);

а.3) значения счетчиков $l = 0$, $m = 0$, $n = 0$, используемых для формирования уникального номера фрейма;

а.4) `type` = `serverHello` = `0x12`;

а.5) `meslen` = `100` — длина сообщения `ServerHelloMessage` в октетах (в сетевом порядке следования октетов сериализованное представление числа 100 имеет вид `0064`).

б) Поскольку результат применения функции $\mathtt{HMAC}_{256}$ занимает 32 октета, то сериализованное представление типа `IntegrityCode`, см. B.2.12, должно занимать 34 октета (два октета `B120` за которыми следует значение имитовставки).

Учитывая это, сервер вычисляет длину дополнения p, см. B.3.1, равную

$$p = 160 - \mathtt{meslen} - 11 - 34 = 15.$$

в) Сервер вырабатывает 15 случайных октетов дополнения

`padding` = `40D48D583153C7DDF7ED494FD189BB`

г) Сервер формирует заголовок и тело фрейма, образованные конкатенацией следующих фрагментов

```
A0
00A0
0000000000
12
0064
521195DEC4E0AF189B94D9EDC0FA915C2FEAC202
32B686D922F0E5FC25299360F0AF056C90280921
184E36FFAB39F9728346388C87659C0209C2B928
9D6A4D277596D26130E33AE93B4E2CE8D6BD8B2F
8E3A3C0ECB203AB839A3A3F09F9E489FD9730400
40D48D583153C7DDF7ED494FD189BB
```

д) Для указанной последовательности октетов сервер вычисляет значение имитовставки, являющееся результатом применения алгоритма $\mathtt{HMAC}_{256}$ на ключе `ePSK`

`B6B42DFAF6C582E6A22BB9EBDD336A05D2FFBBC7A6DA0019A9E05DF0B44EEAC8`.

е) Сервер окончательно формирует тело фрейма, образованное конкатенацией следующих фрагментов

```
A000A0000000000012006 4521195DEC4E0AF189B
94D9EDC0FA915C2FEAC20232B686D922F0E5FC25
299360F0AF056C90280921184E36FFAB39F97283
46388C87659C0209C2B9289D6A4D277596D26130
E33AE93B4E2CE8D6BD8B2F8E3A3C0ECB203AB839
A3A3F09F9E489FD973040040D48D583153C7DDF7
ED494FD189BBB120B6B42DFAF6C582E6A22BB9EB
DD336A05D2FFBBC7A6DA0019A9E05DF0B44EEAC8
```

Д.1.5 Формирование ключевой информации SHTS

Согласно 5.5.1 сервер формирует ключевую информацию SHTK. Для этого, он выполняет следующие шаги:

а) вычисляет общую точку $Q = k_s P_c$

x = B2C9EFB992CCAFBF66056B8CA45B3D50E991015678DB45F2A2CF41074045FE93,
y = 3260346AB3BFE5441A46E7A63E5A43C2B7C011CABC1C639DDB0AD3248B9FB6CF.

б) формирует последовательность октетов R_1, представляющую собой конкатенацию следующих фрагментов

B2C9EFB992CCAFBF66056B8CA45B3D50E991015678DB45F2A2CF41074045FE93
BB769493AF2499C3223648303DB2EFDD568B77843BE77731F4E1539961F8190C

в) формирует последовательность H_1, представляющую собой конкатенацию следующих фрагментов

3020B0B1093132372E302E302E31AF56198E2AA4
124597748DB8382989603D44D7B78224D0F154C6
22C32C85831E05DE277EB89968BBC60B3854283F
855B028B2BDD781A9C3839FC41AD8B8EA32AF243
CA69DCF666C981AD1D7861639A22B20358F4209A
588D2CC94FA2F464FA1ACB00

521195DEC4E0AF189B94D9EDC0FA915C2FEAC202
32B686D922F0E5FC25299360F0AF056C90280921
184E36FFAB39F9728346388C87659C0209C2B928
9D6A4D277596D26130E33AE93B4E2CE8D6BD8B2F
8E3A3C0ECB203AB839A3A3F09F9E489FD9730400

г) вычисляет ключевую информацию

$$\mathtt{SHTS} = \mathtt{HMAC}_{512}(\mathtt{Streebog}_{512}(R_1), \mathtt{Streebog}_{512}(H_1))$$

и ключи eSHTS и iSHTS равенствами

$$\mathtt{eSHTK} = \mathtt{SHTS}[0, \ldots, 31], \quad \mathtt{iSHTK} = \mathtt{SHTS}[32, \ldots, 63];$$

eSHTK:
FF56693FCD85E0AAD0AD3D974682B2A8F0C944315A0F41F06E11A2E7EC2F5C79

iSHTK:
6AC4FFCC2A6363F712C5FDEC8A4A4E26E0CE6B789E0B5BBFFDA8156144C63D27

Д.1.6 Формирование сообщения VerifyMessage

В соответствии с 5.6.2 сервер формирует сообщение VerifyMessage:

а) формирует последовательность H_2, представляющую собой конкатенацию следующих фрагментов[7]

```
3020B0B1093132372E302E302E31AF56198E2AA4
124597748DB8382989603D44D7B78224D0F154C6
22C32C85831E05DE277EB89968BBC60B3854283F
855B028B2BDD781A9C3839FC41AD8B8EA32AF243
CA69DCF666C981AD1D7861639A22B20358F4209A
588D2CC94FA2F464FA1ACB00

521195DEC4E0AF189B94D9EDC0FA915C2FEAC202
32B686D922F0E5FC25299360F0AF056C90280921
184E36FFAB39F9728346388C87659C0209C2B928
9D6A4D277596D26130E33AE93B4E2CE8D6BD8B2F
8E3A3C0ECB203AB839A3A3F09F9E489FD9730400
```

б) вычисляет значение $\mathtt{Streebog}_{512}(H_2)$ и формирует сообщение VerifyMessage, представляющее собой конкатенацию следующих фрагментов

```
B1
10
40D31950C0BA9661D0ECFBDFD489321B
B0
```

в) сервер передает сформированное сообщение VerifyMessage, ключ шифрования eSHTK и ключ имитозащиты iSHTK на транспортный уровень для формирования фрейма.

Д.1.7 Формирование фрейма для сообщения VerifyMessage

а) Для формирования фрейма сервер использует следующие значения:

а.1) tag = 0xA2 — фрейм передается в зашифрованном виде;

а.2) length = 160 — общая длина фрейма составляет 160 октетов (в сетевом порядке следования октетов сериализованное представление числа 160 имеет вид 00A0);

а.3) значения счетчиков $l = 0$, $m = 1$, $n = 0$, используемых для формирования уникального номера фрейма, см. 7.2;

а.4) type = verifyMessage = 0x13;

[7] Поскольку в данном примере сервер не отсылает клиенту сообщения с расширениями, то значение последовательности октетов H_2 совпадает значением последовательности октетов H_1.

а.5) `meslen = 19` — длина сообщения `ServerHelloMessage` в октетах (в сетевом порядке следования октетов сериализованное представление числа 19 имеет вид `0013`).

б) Поскольку результат применения функции выработки имитовставки, регламентируемой ГОСТ Р 34.13-2015 для блочного алгоритма Кузнечик, занимает 16 октетов, то сериализованное представление типа `IntegrityCode`, см. B.2.12, должно занимать 18 октетов (два октета `B110` за которыми следует значение имитовставки).

Учитывая это, сервер вычисляет длину дополнения p, см. B.3.1, равную

$$p = 160 - \texttt{meslen} - 11 - 18 = 112.$$

в) Сервер вырабатывает 112 случайных октетов дополнения

```
5B0BA8823545D1A7262DB97792BB29488DFF05B1
00CF498AE84DE89F9EF0E7AA66ED4B72FF7E94E3
3D5810C4A0D46B8400B0B62C505326C00F471B47
B881AAC0F06879E8CA0279FD89B1AB775BF5DECF
1E2F6D3B1E5B046ED3D35B5BA511F7F1AB6CEFDF
66D9A35089AA85AEB251D362
```

г) Сервер формирует заголовок и тело фрейма, образованные конкатенацией следующих фрагментов

```
A2
00A0
0000010001
13
0013
B11040D31950C0BA9661D0ECFBDFD489321BB0
5B0BA8823545D1A7262DB97792BB29488DFF05B1
00CF498AE84DE89F9EF0E7AA66ED4B72FF7E94E3
3D5810C4A0D46B8400B0B62C505326C00F471B47
B881AAC0F06879E8CA0279FD89B1AB775BF5DECF
1E2F6D3B1E5B046ED3D35B5BA511F7F1AB6CEFDF
66D9A35089AA85AEB251D362
```

д) Для указанной последовательности октетов сервер вычисляет значение имитовставки, являющееся результатом применения алгоритма ГОСТ 34.13-2015 для блочного шифра «Кузнечик» на ключе `iSHTK`

`465312EC4F098E939EA9132378CC3728`.

е) Сервер зашифровывает в режиме гаммирования на на ключе `eSHTK` следующую последовательность октетов (тело фрейма)

130013B11040D31950C0BA9661D0ECFBDFD48932
1BB05B0BA8823545D1A7262DB97792BB29488DFF
05B100CF498AE84DE89F9EF0E7AA66ED4B72FF7E
94E33D5810C4A0D46B8400B0B62C505326C00F47
1B47B881AAC0F06879E8CA0279FD89B1AB775BF5
DECF1E2F6D3B1E5B046ED3D35B5BA511F7F1AB6C
EFDF66D9A35089AA85AEB251D362

В результате зашифрования получается следующий шифртекст

92EC319A8432F8C04ED80018F522BACEF7EE84DF
D8115F5CEF0CB6D34C54D0005485825C2BCB6B2F
1127E1D5007385E6CFDF75811F564A8488BE7FEC
DC5816292EB39E1E234AA0A9039C8C39D78ADA5C
65BA8FD0B30CBDCAD8A9895E827857DA9B2DB0A8
4FE19C03D7F05C5A104FFDE87F17813878863720
072BDE49ACD62C81198E6D5B52D8

ж) Сервер окончательно формирует фрейм, представляющий собой объединение следующих фрагментов

A2
00A0
0000010001
92EC319A8432F8C04ED80018F522BACEF7EE84DF
D8115F5CEF0CB6D34C54D0005485825C2BCB6B2F
1127E1D5007385E6CFDF75811F564A8488BE7FEC
DC5816292EB39E1E234AA0A9039C8C39D78ADA5C
65BA8FD0B30CBDCAD8A9895E827857DA9B2DB0A8
4FE19C03D7F05C5A104FFDE87F17813878863720
072BDE49ACD62C81198E6D5B52D8
B1
10
465312EC4F098E939EA9132378CC3728

Д.1.8 Формирование ключевой информации CHTS

Согласно 5.5.1 клиент формирует ключевую информацию CHTS. Для этого, он выполняет следующие шаги:

а) вычисляет общую точку $Q = k_c P_s$

x = B2C9EFB992CCAFBF66056B8CA45B3D50E991015678DB45F2A2CF41074045FE93,
y = 3260346AB3BFE5441A46E7A63E5A43C2B7C011CABC1C639DDB0AD3248B9FB6CF.

б) формирует последовательность октетов R_1, представляющую собой конкатенацию следующих фрагментов

B2C9EFB992CCAFBF66056B8CA45B3D50E991015678DB45F2A2CF41074045FE93
BB769493AF2499C3223648303DB2EFDD568B77843BE77731F4E1539961F8190C

в) формирует последовательность H_3, представляющую собой конкатенацию следующих фрагментов

3020B0B1093132372E302E302E31AF56198E2AA4
124597748DB8382989603D44D7B78224D0F154C6
22C32C85831E05DE277EB89968BBC60B3854283F
855B028B2BDD781A9C3839FC41AD8B8EA32AF243
CA69DCF666C981AD1D7861639A22B20358F4209A
588D2CC94FA2F464FA1ACB00

521195DEC4E0AF189B94D9EDC0FA915C2FEAC202
32B686D922F0E5FC25299360F0AF056C90280921
184E36FFAB39F9728346388C87659C0209C2B928
9D6A4D277596D26130E33AE93B4E2CE8D6BD8B2F
8E3A3C0ECB203AB839A3A3F09F9E489FD9730400

B11040D31950C0BA9661D0ECFBDFD489321BB0

г) вычисляет ключевую информацию

$$\texttt{CHTS} = \texttt{HMAC}_{512}(\texttt{Streebog}_{512}(R_1), \texttt{Streebog}_{512}(H_3))$$

и ключи eCHTS и iCHTS равенствами

$$\texttt{eCHTK} = \texttt{CHTS}[0, \ldots, 31], \quad \texttt{iCHTK} = \texttt{CHTS}[32, \ldots, 63];$$

eCHTK:
2EC4CCACE18D9D6FC4314E863611876A7BAD9B6C93090FF64D1C275B6E7C9627

iCHTK:
8BC191E3C70D0FAC0A5ED4D873501F4D6773CCC959487FCE9E49B08678FF5F8B

Д.1.9 Формирование сообщения VerifyMessage

В соответствии с 5.6.3 клиент формирует сообщение VerifyMessage:

а) формирует последовательность H_4, представляющую собой конкатенацию следующих фрагментов[8]

3020B0B1093132372E302E302E31AF56198E2AA4
124597748DB8382989603D44D7B78224D0F154C6
22C32C85831E05DE277EB89968BBC60B3854283F

[8]Поскольку в данном примере клиент не отсылает серверу сообщения с расширениями, то значение последовательности октетов H_4 совпадает значением последовательности октетов H_3.

```
855B028B2BDD781A9C3839FC41AD8B8EA32AF243
CA69DCF666C981AD1D7861639A22B20358F4209A
588D2CC94FA2F464FA1ACB00

521195DEC4E0AF189B94D9EDC0FA915C2FEAC202
32B686D922F0E5FC25299360F0AF056C90280921
184E36FFAB39F9728346388C87659C0209C2B928
9D6A4D277596D26130E33AE93B4E2CE8D6BD8B2F
8E3A3C0ECB203AB839A3A3F09F9E489FD9730400

B11040D31950C0BA9661D0ECFBDFD489321BB0
```

б) вычисляет значение $\texttt{Streebog}_{512}(H_4)$ и формирует сообщение `VerifyMessage`, представляющее собой конкатенацию следующих фрагментов

```
B1
10
D4B715576F37104BCA18C4717E9B349A
B0
```

в) сервер передает сформированное сообщение `VerifyMessage`, ключ шифрования `eCHTK` и ключ имитозащиты `iCHTK` на транспортный уровень для формирования фрейма.

Д.1.10 Формирование фрейма для сообщения `VerifyMessage`

а) Для формирования фрейма клиент использует следующие значения:

а.1) `tag` = `0xA2` — фрейм передается в зашифрованном виде;

а.2) `length` = `160` — общая длина фрейма составляет 160 октетов (в сетевом порядке следования октетов сериализованное представление числа 160 имеет вид `00A0`);

а.3) значения счетчиков $l = 0$, $m = 1$, $n = 0$, используемых для формирования уникального номера фрейма, см. 7.2;

а.4) `type` = `verifyMessage` = `0x13`;

а.5) `meslen` = `19` — длина сообщения `ServerHelloMessage` в октетах (в сетевом порядке следования октетов сериализованное представление числа 19 имеет вид `0013`).

б) Поскольку результат применения функции выработки имитовставки, регламентируемой ГОСТ Р 34.13-2015 для блочного алгоритма Кузнечик, занимает 16 октетов, то сериализованное представление типа `IntegrityCode`, см. В.2.12, должно занимать 18 октетов (два октета `B110` за которыми следует значение имитовставки).

Учитывая это, клиент вычисляет длину дополнения p, см. В.3.1, равную

$$p = 160 - \texttt{meslen} - 11 - 18 = 112.$$

в) Сервер вырабатывает 112 случайных октетов дополнения

```
47B2AA5C0B4DB93734F491E04821433405C66FF3
2C4C1EB30A5F16BF7EBFEB47130B32A59360A390
00FB94C21D5E15B1B0F7E48944AA65D331C26C18
FCD81EC37096DE2B74720A54D16FC2B3EBE2A0EA
85297F86807337C79B7E5D5733530610030B84B9
E48B26C88C4AFB2051088ED2
```

г) Клиент формирует заголовок и тело фрейма, образованные конкатенацией следующих фрагментов

```
A2
00A0
0000010001
13
0013
B110D4B715576F37104BCA18C4717E9B349AB0
47B2AA5C0B4DB93734F491E04821433405C66FF3
2C4C1EB30A5F16BF7EBFEB47130B32A59360A390
00FB94C21D5E15B1B0F7E48944AA65D331C26C18
FCD81EC37096DE2B74720A54D16FC2B3EBE2A0EA
85297F86807337C79B7E5D5733530610030B84B9
E48B26C88C4AFB2051088ED2
```

д) Для указанной последовательности октетов клиент вычисляет значение имитовставки, являющееся результатом применения алгоритма ГОСТ 34.13-2015 для блочного шифра «Кузнечик» на ключе `iCHTK`

```
142A2B62038F5AA832DC50E319395659.
```

е) Клиент зашифровывает в режиме гаммирования на на ключе `eCHTK` следующую последовательность октетов (тело фрейма)

```
130013B110D4B715576F37104BCA18C4717E9B34
9AB047B2AA5C0B4DB93734F491E04821433405C6
6FF32C4C1EB30A5F16BF7EBFEB47130B32A59360
A39000FB94C21D5E15B1B0F7E48944AA65D331C2
6C18FCD81EC37096DE2B74720A54D16FC2B3EBE2
A0EA85297F86807337C79B7E5D5733530610030B
84B9E48B26C88C4AFB2051088ED2
```

В результате зашифрования получается следующий шифртекст

```
55CE5B573C6FBE33A128796BEBBE6DCC330450F4
11C26952C90911E65D2E4F22D7DFF06C7C7F4101
FB678BA8BB97ED68B24B475860A9C7BA80D71E10
```

```
D2621E267BBDCBCCB6BB5B7E3039AE606B01ECF9
A3B1B09F56E79917B37C1F8C6E27D30C68AED444
D0C4A6CEDDB3D5AC5888B221E0B17CCB7CD6EE45
A00169D96D021B59FA66995D4D2B
```

ж) Клиент окончательно формирует фрейм, представляющий собой объединение следующих фрагментов

```
A2
00A0
0000010001
55CE5B573C6FBE33A128796BEBBE6DCC330450F4
11C26952C90911E65D2E4F22D7DFF06C7C7F4101
FB678BA8BB97ED68B24B475860A9C7BA80D71E10
D2621E267BBDCBCCB6BB5B7E3039AE606B01ECF9
A3B1B09F56E79917B37C1F8C6E27D30C68AED444
D0C4A6CEDDB3D5AC5888B221E0B17CCB7CD6EE45
A00169D96D021B59FA66995D4D2B
B1
10
142A2B62038F5AA832DC50E319395659
```

Д.1.11 Формирование ключевой информации CATS и SATS

Согласно 5.5.2 клиент и сервер независимо формируют ключевую информацию CATS и SATS. Для этого они выполняют следующие шаги:

а) формируют последовательность октетов R_2, представляющую собой конкатенацию следующих фрагментов

```
7365727665722D303031
636C69656E742D3030313337
BB769493AF2499C3223648303DB2EFDD568B77843BE77731F4E1539961F8190C
```

б) формируют последовательность H_5, представляющую собой конкатенацию следующих фрагментов

```
3020B0B1093132372E302E302E31AF56198E2AA4
124597748DB8382989603D44D7B78224D0F154C6
22C32C85831E05DE277EB89968BBC60B3854283F
855B028B2BDD781A9C3839FC41AD8B8EA32AF243
CA69DCF666C981AD1D7861639A22B20358F4209A
588D2CC94FA2F464FA1ACB00

521195DEC4E0AF189B94D9EDC0FA915C2FEAC202
32B686D922F0E5FC25299360F0AF056C90280921
184E36FFAB39F9728346388C87659C0209C2B928
```

9D6A4D277596D26130E33AE93B4E2CE8D6BD8B2F
8E3A3C0ECB203AB839A3A3F09F9E489FD9730400

B11040D31950C0BA9661D0ECFBDFD489321BB0

B110D4B715576F37104BCA18C4717E9B349AB0

в) вычисляют ключевое значение $T = \mathtt{HMAC}_{512}(x(Q), R_2)$

184E36FFAB39F9728346388C87659C0209C2B928D3B4EDCEEBFF1642BFF0C1C8
DBDE406A24297CF8D56C7F0162CB8EA76F88AE3A30D1587040D2D30F7C08D05F

г) вычисляют последовательность октетов $A_0 = \mathtt{Streebog}_{512}(H_5)$

76F37104BCA18C47184E36FFAB39F9728346388C87659C0209C2B928D3B4EDCEE
855B028B2BDD7BFF1642BFF0C1C822C32C85831E05DE277EB89968BBC60B38542

д) вычисляют последовательность октетов $A_1 = \mathtt{HMAC}_{512}(T, A_0)$

FD952888AB4C28BAEA62979CBD3A2225F5449E97E44DC8118378651F037BF81DD
F0A7236A1FE9B5635096681216CEB50303793E839AAFE87DCE1BC434A5E4D56AF

е) вычисляют последовательность октетов $\mathtt{CATS} = \mathtt{HMAC}_{512}(T, A_1||A_0)$

D3B4EDCEEBFF1642BFF0C1C8DBDE406A24297CF8D56C7F0162CB8EA76F88AE3A
30D1587040D2D30F7C08D05FDD6B9080DC0495985A5579C8EFAAE4A247C8578B

ж) вычисляют последовательность октетов $A_2 = \mathtt{HMAC}_{512}(T, A_1)$

ED29E6770DCEAF1750CBBED1FCF09439FFB0225EC618A74F55C9C44B2E081C47
DC377BFEFA9E89B2BDEEDD941804994B3BEDBAF8E226453D19444CD504BDC4F4

з) вычисляют последовательность октетов $\mathtt{SATS} = \mathtt{HMAC}_{512}(T, A_2||A_0)$

B33364AB578B1579EEB346906B921102CBD9455AB821F02BCF2C4ACEE5DDA3AA
F37D04D75AC839C709ABD63111B60109DE0CCCE77E95EBC89662CDE9AE10BA23

Значения ключевой информации CATS и SATS передаются в протокол прикладной передачи данных для обеспечения конфиденциальности и целостности передаваемой информации.

Библиография

[1] Р 1323565.1.012.-2017 Информационная технология. Криптографическая защита информации. Принципы разработки и модернизации шифровальных (криптографических) средств защиты информации

[2] Р 50.1.114 – 2016 Информационная технология. Криптографическая защита информации. Параметры эллиптических кривых для криптографических алгоритмов и протоколов

[3] ISO/IEC 9899:1999 Programming languages – C

[4] Российские рекомендации по использованию сертификатов в формате x509.

[5] Российские рекомендации по использованию сертификатов в контрольных и измерительных устройствах.

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

[7] Р 1323565.1.xxx Информационная технология. Криптографическая защита информации. Режим работы блочных шифров, осуществляющий одновременно шифрование и аутентификацию (проект).

[8] Popov V., Kurepkin I., Leontiev S. Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms. – RFC4357. – 2006.

[9] Р 1323565.1.004 – 2017 Информационная технология. Криптографическая защита информации. Схемы выработки общего ключа с аутентификацией на основе открытого ключа.

[10] 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

[11] *Krawczyk H.* HMAC-based Extract-and-Expand Key Derivation Function (HKDF). – RFC 5869. – 2010.

[12] Р 1323565.1.017 – 2018 Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов блочного шифрования.

[13] Р 1323565.1.021 – 2018 Информационная технология. Криптографическая защита информации. Рекомендации по стандартизации. 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.

[14] *Blom R.* Nonpublic key distribution//Advances in Cryptology. – Proceedings of EUROCRYPT ’82. – Plenum. New York. – 1983. – pp. 231–236.

[15] Приказ ФСБ России от 09 февраля 2005 г. №. 66 (в редакции приказа ФСБ России от 12 апреля 2010 г. №. 173). Об утверждении положения о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации (Положение ПКЗ – 2005).