МАТЕМАТИЧЕСКИЕ МЕТОДЫ КРИПТОГРАФИИ

УДК 003.26 + 004.056

ОСНОВНЫЕ ЭТАПЫ РАЗВИТИЯ КРИПТОГРАФИЧЕСКИХ ПРОТОКОЛОВ SSL/TLS И IPsec

И. В. Мартыненков

Астраханский государственный технический университет, г. Астрахань, Россия

Рассматриваются основные этапы развития криптографических протоколов от SSL 2.0 (Secure Socket Layer) до TLS 1.3 (Transport Layer Security), обеспечивающих защиту данных транспортного уровня модели OSI. Приводится краткое описание модификации протокола RuTLS, построенного на базе TLS 1.3, и их основные отличия. Развитие IPsec, предоставляющего криптографическую защиту коммуникаций на сетевом уровне модели OSI, рассмотрено на примерах развития трёх наиболее часто применяемых протоколов, на основе которых он строится. В их число входят IKE (Internet Key Exchange), AH (Authentication Header), ESP (Encapsulation Security Payload).

Ключевые слова: *криптографические протоколы, SSL, TLS, IPsec.*

DOI 10.17223/20710410/51/2

THE MAIN STAGES OF DEVELOPMENT OF THE CRYPTOGRAPHIC PROTOCOLS SSL/TLS AND IPsec

I. V. Martynenkov

Astrakhan State Technical University, Astrakhan, Russia

E-mail: mivpost@yandex.ru

The paper discusses the main stages of development of cryptographic protocols from SSL 2.0 (Secure Socket Layer) to TLS 1.3 (Transport Layer Security), which ensure the protection of transport layer data in the OSI model. A brief description of the modification of the RuTLS protocol based on TLS 1.3 and their main differences is given. The development of IPsec, which provides cryptographic protection of communications at the network level of the OSI model, is considered using examples of the development of the three most commonly used protocols. These include IKE (Internet Key Exchange), AH (Authentication Header), and ESP (Encapsulation Security Payload). For the SSL/TLS and IPsec specifications, the basic handshake protocols and the main stages of their development are considered. The described handshakes include primary cryptographic information exchange cycles in the form of identifiers of interaction participants, one-time numbers, lists of supported cryptographic combinations. Authentication of participants based on certificates, shared symmetric keys, data exchange for establishing a shared Diffie — Hellman secret, development of key material for secret keys of communication sessions, message authentication, and other cryptographic parameters are presented. For different versions of SSL/TLS and IPsec,

the logical structures of application data cryptographic protection functions are described.

Keywords: *cryptographic protocols, SSL, TLS, IPsec.*

Введение

Целью работы является описание основных этапов развития криптографических протоколов защиты транспортного и сетевого уровней модели OSI, представленных семейством спецификаций SSL/TLS, а также IPsec. Рассматриваются разные этапы взаимодействия участников информационного обмена, необходимые для установления безопасного канала связи. К ним относятся, например, обмен первичной криптографической информацией в виде идентификаторов участников взаимодействия, одноразовых номеров, списков поддерживаемых комбинаций криптографических алгоритмов; аутентификация участников на основе сертификатов, общих симметричных ключей; обмен данными для установления разделяемого секрета Диффи — Хеллмана; выработка ключевого материала для секретных ключей сеансов связи, аутентификации сообщений и другие криптографические параметры. Дополнительно приводятся основные меры повышения безопасности протокола RuTLS, разработанного на основе TLS 1.3.

В связи с непрерывным анализом и исследованиями криптографических протоколов в рамках предотвращения возникновения выявленных угроз безопасности с течением времени способы взаимодействия участников информационного обмена развивались и претерпевали изменения. Развитие криптографических протоколов в направлении увеличения безопасности защищённых соединений рассмотрено согласно хронологическому порядку выпуска соответствующих спецификаций от SSL 2.0 до TLS 1.3, а также спецификаций, составляющих основу функционала протокола IPsec, а именно IKE, AH/ESP. При этом приводятся только наиболее важные изменения относительно предыдущих релизов криптографических протоколов.

1. Развитие криптографических протоколов семейства SSL/TLS

1.1. Протокол SSL 2.0

Протокол SSL 2.0 [1] разработан компанией «Netscape» и в 1995 г. опубликован в качестве исторической справки как первый и небезопасный [2] представитель семейства одноимённых протоколов.

Рукопожатия SSL 2.0 включают три типовых сценария: установка нового безопасного соединения, возобновление соединения и аутентификация клиента C (Client). Сообщения квитирования установки нового соединения состоят из шести циклов. Клиент отправляет случайное число r_c (Random, размер r от 16 до 32 байт), генерируемое для каждого соединения в рамках аутентификации и борьбы с атаками воспроизведения (Replay Attack), а также список криптонаборов клиента cs_c (Cipher Suites). Сервер S (Server) отвечает случайным числом идентификатора соединения cid_s (Connection Identifier, размер cid от 16 до 32 байт), генерируемым для целей, аналогичных r_c, а также сертификатом crt_s (Certificate) открытого ключа сервера (e_s) типа X.509 с подписью согласно PKCS#1 и выбранным криптонабором cs_s.

Клиент генерирует главный секрет mk (Master Key) и передаёт его в зашифрованном виде с использованием открытого ключа сервера e_s. Клиент и сервер применяют согласованный mk для генерации k_1, k_2 и iv (сеансовые ключи записи клиента — чтения сервера, чтения клиента — записи сервера, которые совпадают с ключами аутен-

тификации сообщений, и вектор инициализации для блочных шифров) за счёт представленного ниже преобразования, константа зависит от согласованного криптонабора (символ «$\|$» означает конкатенацию данных):

$$(k_1 \parallel k_2 \parallel iv) = \mathrm{MD5}(mk \parallel \mathrm{const} \parallel r \parallel cid).$$

Затем на согласованных криптонаборах и (выработанных) ключах происходит последовательный обмен идентификаторами cid_s, r_c и выработанным сервером новым идентификатором сеанса sid_s (Session Identifier, 16 байт), который содержит копию mk. Полученный sid_s может храниться в кэш-памяти сервера и клиента для целей будущего ускоренного восстановления сеанса:

$$\begin{aligned}
&1) \quad C \to S: \quad r_c, cs_c;\\
&2) \quad C \leftarrow S: \quad cid_s, crt_s, cs_s;\\
&3) \quad C \to S: \quad E_{e_s}(mk);\\
&4) \quad C \to S: \quad E_{k_1}(cid_s);\\
&5) \quad C \leftarrow S: \quad E_{k_2}(r_c);\\
&6) \quad C \leftarrow S: \quad E_{k_1}(sid_s).
\end{aligned}$$

Для попытки возобновления сессии циклы 1, 2 и 3 заменяются двумя другими циклами, в которых флаг sid_{hc} указывает на возможность возобновления сессии при наличии $sid_s = sid_c$ в кэш-памяти сервера, сообщения передаются в открытом виде. Применение cid и sid делают сеансовые ключи зависимыми от восстанавливаемого и текущего сеанса:

$$\begin{aligned}
&1') \quad C \to S: \quad r_c, sid_c, cs_c;\\
&2') \quad C \leftarrow S: \quad cid_s, sid_{hc}.
\end{aligned}$$

Аутентификация клиента происходит за счёт дополнительного выбора сервером типа аутентификации aut_s (Authentication), типа сертификата клиента crt_t (Certificate Type) и данных rd_c (Response Data) как функции от aut_s между циклами 5 и 6, сообщения зашифрованы с использованием согласованных криптонаборов и ключей:

$$\begin{aligned}
&5'') \quad C \leftarrow S: \quad E_{k_2}(aut_s, r'_s);\\
&6'') \quad C \to S: \quad E_{k_1}(crt_t, crt_c, rd_c).
\end{aligned}$$

После выработки общих параметров безопасности происходит обмен данными приложения $data$, которые защищены в порядке преобразования MAC-and-Encrypt (M&E). Сообщения строятся с использованием симметричных ключей шифрования k, ключей аутентификации сообщений k, дополнения pad до кратности размера блока блочного шифра, а также 32-разрядного порядкового номера SN (Sequence Number) сообщения (инкрементный счетчик с зацикливанием):

$$data_{\mathrm{app}} = E_k(data \parallel pad) \parallel \mathrm{MD5}(k \parallel data \parallel pad \parallel SN).$$

Для SSL 2.0 определены криптонаборы из комбинаций RC2, IDEA, DES, режима CBC, MD5, RSA и сертификатов X.509. Экспортные реализации передают часть ключа в открытом виде. Например, для экспортного RC4 88 бит ключа передаются открыто и только 40 бит в зашифрованном виде.

SSL 2.0 поддерживает оповещения о следующих ошибках: отсутствие выбранного шифронабора/ключа, отсутствие сертификата клиента при его аутентификации, некорректный сертификат клиента с ошибочной подписью или идентификаторами, неподдерживаемый тип сертификата.

1.2. Протокол SSL 3.0

Протокол SSL 3.0 [3] также неустойчив ко многим атакам, однако по сравнению с SSL 2.0 претерпел значительные изменения и стал основоположником будущих версий TLS. Версия SSL 3.0 устанавливается в значение 0x0300. Протокол квитирования включает пять основных циклов. Клиент отправляет открытый запрос на сервер с самой новой версией поддерживаемого протокола v_c, случайным числом r_c (32 байта: 4 байта время/дата в UNIX-формате и 28 байт генератора случайных чисел, $r_c \neq r_s$), $sid_c = 0$ для новой сессии, список поддерживаемых криптонаборов cs_c и методов сжатия $comp_c$ (Compression). Сервер отвечает своими параметрами и избранными cs_s и $comp_s$ из списков клиента.

При установлении соединения могут использоваться опциональные сообщения (отмечены индексом «*»). К ним относятся сертификат сервера crt_s, типы сертификатов и информация о центрах сертификации (crt_{req}) неанонимного сервера, сообщения ke_s (Key Exchange), содержащие параметры prm_s (Parameters) для обмена ключевой информацией: модуль ($\bmod p$), открытая фиксированная или временная экспонента (e_s) RSA; модуль ($\bmod p$), генератор (g) и открытое значение (g^x) Диффи — Хеллмана (DH); случайное число (r_f) Fortezza.

Открытые параметры подписываются RSA или DSS. Подписываемое RSA сообщение кодируется в формате блока типа 1 PKCS#1 [5], шифрование RSA выполняется над сообщением, отформатированным в виде блока типа 2 PKCS#1:

$$sig_{\mathrm{RSA}} = E_{\mathrm{RSA}}(\mathrm{MD5}(r_c \parallel r_s \parallel prm_s \parallel \mathrm{SHA1}(r_c \parallel r_s \parallel prm_s))),$$
$$sig_{\mathrm{DSS}} = E_{\mathrm{DSS}}(\mathrm{SHA1}(r_c \parallel r_s \parallel prm_s)).$$

Клиент отвечает зашифрованным предварительным главным секретом $prems$ (Pre-Master Secret). Для DH $prems$ является согласованным ключом DH. При аутентификации и согласовании ключей на основе RSA 48 байт $prems = (v_c \parallel rnd_c)$, где v_c (Version) — 2 байта, rnd_c (Random Data) — 46 случайных байт. Сообщение зашифровывается на открытом ключе сервера e_s из сертификата сервера crt_s или временном ключе e_t из обмена сервера ke_s. Для защиты $prems$ в случае алгоритма Fortezza также передаётся открытый ключ алгоритма обмена ключами (KEA) и его подпись на секретном ключе DSS клиента, 128 случайных байт для расчёта KEA, симметричные ключи записи сервера и клиента, зашифрованные с использованием токена ключа шифрования (TEK), векторы инициализации для шифрования сервера, клиента и TEK:

$$ke_c = E_{\mathrm{RSA}}(prems).$$

Далее формируется 48 байт главного секрета ms (Master Secret) и блок ключевого материала $keyb$ (Key Block), который последовательно нарезается на ключи необходимого размера в следующем порядке: ключи записи MAC-кодов клиента/сервера, ключи записи симметричного шифрования клиента/сервера, векторы инициализации клиента/сервера. Ключи экспортных реализаций строятся хешированием ключей и случайных значений из этапа генерации $keyb$. Для Fortezza главный секрет ms используется только для вычисления MAC-кодов:

$$ms = \mathrm{MD5}(prems \parallel \mathrm{SHA}(\text{«}A\text{»} \parallel prems \parallel r_c \parallel r_s)) \parallel \mathrm{MD5}(prems \parallel$$
$$\parallel \mathrm{SHA}(\text{«}BB\text{»} \parallel prems \parallel r_c \parallel r_s)) \parallel \mathrm{MD5}(prems \parallel \mathrm{SHA}(\text{«}CCC\text{»} \parallel prems \parallel r_c \parallel r_s));$$

$$keyb = (k_{c_w}^{\mathrm{MAC}} \parallel k_{s_w}^{\mathrm{MAC}} \parallel k_{c_w} \parallel k_{s_w} \parallel iv_c \parallel iv_s) =$$
$$= \mathrm{MD5}(ms \parallel \mathrm{SHA}(\text{«}A\text{»} \parallel ms \parallel r_s \parallel r_c)) \parallel \mathrm{MD5}(ms \parallel \mathrm{SHA}(\text{«}BB\text{»} \parallel ms \parallel r_s \parallel r_c)) \parallel$$
$$\parallel \mathrm{MD5}(ms \parallel \mathrm{SHA}(\text{«}CCC\text{»} \parallel ms \parallel r_s \parallel r_c)) \parallel \ldots;$$

$$k^e_{c_w} = \mathrm{MD5}(k_{c_w} \parallel r_c \parallel r_s), \quad k^e_{s_w} = \mathrm{MD5}(k_{s_w} \parallel r_s \parallel r_c),$$
$$iv^e_c = \mathrm{MD5}(r_c \parallel r_s), \quad iv^e_s = \mathrm{MD5}(r_s \parallel r_c).$$

Отправляется также финальное сообщение, которое является первым защищённым сообщением на согласованных секретах и криптонаборах cs. Используется согласованный хеш H (Hash, MD5 или SHA), const = 0x434C4E54/0x53525652 (клиент/сервер), $hsmsg$ (Handshake Message) — все сообщения квитирования, исключая текущие fin (Finished) и cs, pad_1 (Padding, 48 значений 0x36 для MD5 или 40 значений для SHA) и pad_2 (48 значений 0x5C для MD5 или 40 значений для SHA):

$$fin_c = H(ms \parallel pad_2 \parallel H(hsmsg \parallel \mathrm{const} \parallel ms \parallel pad_1)).$$

Верификация сертификата (crt_{verif}) содержит главный секрет ms и все сообщения квитирования, исключая crt_{verif} и cs. Данные подписываются алгоритмом RSA или DSS на секретном ключе сертификата клиента, что необходимо для явной проверки сертификата клиента с возможностью подписи:

$$sig_{\mathrm{RSA/DSS}} = E_{\mathrm{RSA\ /\ DSS}}(H(ms \parallel pad_2 \parallel H(hsmsg \parallel ms \parallel pad_1))).$$

В заключительном цикле сервер формирует fin_s, которое включает все предыдущие сообщения квитирования, причём $fin_c \neq fin_s$, так как, как минимум, fin_c не содержит fin_s:

$$\begin{aligned}
&1) \quad C \to S: \quad v_c, r_c, sid_c, cs_c, comp_c;\\
&2) \quad C \leftarrow S: \quad v_s, r_s, sid_s, cs_s, comp_s;\\
&3) \quad C \leftarrow S: \quad crt^*_s, ke^*_s, crt^*_{req};\\
&4) \quad C \to S: \quad ke_c, cs_c, fin_c, crt^*_c, crt^*_{\mathrm{verif}};\\
&5) \quad C \leftarrow S: \quad cs_s, fin_s.
\end{aligned}$$

Для восстановления сессии циклы 1–5 заменяются тремя обменами, причем сервер S ищет запрашиваемый sid_c (32 байта) в кэш-памяти. Это позволяет создавать несколько независимых соединений без полного квитирования на уже согласованных криптографических параметрах, экономя дорогостоящие операции асимметричной криптографии. Аутентификация sid_c выполняется в финальных сообщениях fin:

$$\begin{aligned}
&1') \quad C \to S: \quad sid_c;\\
&2') \quad C \leftarrow S: \quad sid_s, E_{k_{s_w}}(cs_c), fin_s;\\
&3') \quad C \to S: \quad E_{k_{c_w}}(cs_c), fin_c.
\end{aligned}$$

Данные приложения защищены в порядке преобразования MAC-then-Encrypt (MtE). Используется функция сжатия $comp$ и её идентификатор ($comp_t$), 64-разрядный порядковый номер сообщения SN (по два счётчика на приём/передачу для S и C, обнуляемых при изменении шифронаборов), а также размер сжатых данных $\mathrm{comp}(data)_{\mathrm{len}}$. Для поточных шифров используется структура функции, приведенная ниже, в которой ключевая гамма распространяется на следующие блоки данных непрерывно. Для блочных шифров вводится дополнение (pad) и его длина (pad_{len}) в конец блока открытого текста. В режиме CBC вектор инициализации первого блока вырабатывается протоколом квитирования, для последующих блоков iv равен последнему блоку предыдущего шифртекста:

$$\begin{aligned}
data_{\mathrm{app}} = E_{k_{c/s_w}}(comp(data) \parallel H(k^{\mathrm{MAC}}_{c/s_w} \parallel pad_2 \parallel H(k^{\mathrm{MAC}}_{c/s_w} \parallel pad_1 \parallel SN \parallel\\
\parallel comp_t \parallel comp(data)_{\mathrm{len}} \parallel comp(data)) \parallel (pad \parallel pad_{\mathrm{len}})^*)).
\end{aligned} \tag{1}$$

Криптонаборы SSL 3.0 содержат алгоритмы шифрования RC2-40, IDEA, DES-40, DES, 3DES-EDE и Fortezza в режиме CBC, RC4-40/128. Обмен ключами и аутентификация реализуются на основе алгоритма RSA, эфемерного DHE, анонимного DH и статического DH с подписями DSS и RSA и их экспортными реализациями, а также алгоритмом Fortezza-KEA. Целостность обеспечивается функциями MD5 и SHA.

Для SSL 3.0 определено 12 типов оповещений об ошибках с двумя уровнями опасности: предупреждение и фатальная ошибка. Проверки затрагивают корректность MAC-кодов, сообщений рукопожатия, сертификатов и других криптографических параметров сервера и клиента.

1.3. Протокол TLS 1.0

По сравнению с SSL 3.0 протокол TLS 1.0 [4] добавляет опциональные данные расширений ext (Extensions) в первом цикле, в том числе для совместимости с будущими версиями TLS. Они включаются в дайджесты рукопожатия. Расширения состоят из типа расширения и данных расширения ($ext = ext_t \parallel ext_{data}$) и могут включать имя сервера, длину максимального фрагмента, сетевой адрес сертификата клиента, доверенные ключи корневого центра сертификации, усечённый HMAC или статус запроса. Остальные циклы полного квитирования и восстановления сессий архитектурно изменений не претерпели и внешне соответствуют SSL 3.0, однако изменены способы вычисления их сообщений. Версия TLS 1.0 устанавливается в значение 0x0301:

$$1)\quad C \to S:\ \ v_c, r_c, sid_c, cs_c, comp_c, ext.$$

Взамен обычного хеширования TLS 1.0 определяет новую функцию вычисления случайных данных PRF (Pseudo-Random Function), которая применяет HMAC на основе MD5 и SHA1. На вход PRF поступает секрет (Secret) $s = (s_1 \parallel s_2)$, состоящий из частей одинаковой длины. Для нечётных длин s последний байт s_1 совпадает с первым байтом s_2. Используются также начальное заполнение $seed$ и метка-идентификатор L (Label), дополнительно вычисляются параметры вида $a_0 = seed$, $a_i = \text{HMAC-H}(s_j, a_{i-1})$, $i = 1, 2, \ldots$, причём для HMAC-MD5 и HMAC-SHA1 количество итераций независимо и может отличаться, части суммируются по модулю 2, а избыточные данные отбрасываются:

$$\begin{gathered}\text{PRF}(s, L, seed) = \\ = \text{HMAC-MD5}(s_1, a_1 \parallel L \parallel seed) \parallel \ldots \parallel \text{HMAC-MD5}(s_1, a_i \parallel L \parallel seed) \oplus \\ \oplus\ \text{HMAC-SHA1}(s_2, a_1 \parallel L \parallel seed) \parallel \ldots \parallel \text{HMAC-SHA1}(s_2, a_i \parallel L \parallel seed).\end{gathered} \tag{2}$$

Протокол TLS 1.0 поддерживает сертификаты X.509 [5] и исключает алгоритм Fortezza в сообщениях обмена ключевой информацией, в остальном ke_c и ke_s соответствуют SSL 3.0. Криптографические преобразования для получения 48 байт главного секрета ms, блока ключевого материала сеансовых ключей $keyb$ и векторов инициализации iv, включая экспортируемые данные, в TLS 1.0 применяют PRF (2) с константными строками на входе. Однако для вычисления $prems$ явно задаётся использование версии протокола клиента v_c из первого цикла:

$$\begin{gathered}ms = \text{PRF}(prems, \text{«master secret»}, r_c \parallel r_s), \\ keyb = (k^{\text{MAC}}_{c_w} \parallel k^{\text{MAC}}_{s_w} \parallel k_{c_w} \parallel k_{s_w} \parallel iv_c \parallel iv_s) = \text{PRF}(ms, \text{«key expansion»}, r_s \parallel r_c), \\ k^e_{c_w} = \text{PRF}(k_{c_w}, \text{«client write key»}, r_c \parallel r_s), \\ k^e_{s_w} = \text{PRF}(k_{s_w}, \text{«server write key»}, r_c \parallel r_s), \\ ivblock = (iv^e_c \parallel iv^e_s) = \text{PRF}(\text{«»}, \text{«IV block»}, r_c \parallel r_s).\end{gathered} \tag{3}$$

Финальные сообщения в TLS 1.0 теперь применяют PRF (2) с константной строкой, идентифицирующей отправителя. Исключая текущее, они включают все сообщения протокола квитирования и составляют 12 байт:

$$fin_{c/s} = \mathrm{PRF}(ms, \text{«client/server finished»}, \mathrm{MD5}(hsmsg) \parallel \mathrm{SHA1}(hsmsg)). \qquad (4)$$

Для явной проверки сертификата клиента SSL 3.0 применял двойное хеширование. TLS 1.0 упрощает вычисления crt_{verif}, в которых RSA или DSS подписывают данные вида $H(hsmsg)$, $hsmsg$ — также все сообщения квитирования, исключая crt_{verif} и cs.

Протокол TLS 1.0 исключает поддержку Fortezza для согласования общего секрета и симметричного шифрования, однако в остальном криптонаборы соответствуют SSL 3.0. Порядок криптографических преобразований данных приложения в TLS 1.0 соответствует (1) протокола SSL 3.0, однако в них функции хешировании MD5 и SHA1 заменены на HMAC-MD5 и HMAC-SHA1, а также исключены дополнения. TLS 1.0 предписывает поддержку обязательного криптографического набора в виде TLS-DHE-DSS-WITH-3DES-EDE-CBC-SHA.

Оповещения об ошибках TLS 1.0 расширены включением проверок корректности расшифровки, достоверности подписей, финальных сообщений, сообщений обмена ключами. Добавлен контроль переполнения длин сообщений, наличия доверенного корневого центра сертификации, контроль допустимых значений полей сообщений и другие. Всего 23 типа сообщений об ошибках.

1.4. Протокол TLS 1.1

Протокол TLS 1.1 [6] оставляет многие функции и способы вычисления сообщений аналогичными TLS 1.0. К ним относятся функция PRF, сообщения согласования общих секретов ke, финальные сообщения fin, сообщения верификации сертификата клиента crt_{verif}, порядок вычисления ms и ключевого материала $keyb$. Версия TLS 1.1 устанавливается в значение 0x0302. Тем не менее параметры TLS 1.1 теперь задокументированы в реестре IANA. Для алгоритма RSA введена поддержка новой кодировки сообщений, направленной на защиту от атак Блейхенбахера [7], включены сертификаты X.509v3 [8].

Порядок криптографических преобразований в TLS 1.1 при шифровании данных приложения остается MtE. Функция поточного шифрования совпадает с функцией TLS 1.0. Однако, в отличие от предыдущих версий протокола, в целях предотвращения атак, описанных в [9], TLS 1.1 вводит явный вектор инициализации iv в единственном поддерживаемом режиме блочного шифрования CBC. После генерации вектор инициализации передается в зашифрованном виде как первый блок шифртекста:

$$iv = r \oplus mask,$$
$$data_{\mathrm{app}} = E_{k_w}(iv \parallel comp(data) \parallel \text{HMAC-H}(\ldots)).$$

Значение $mask$ соответствует 0 или предыдущему CBC-блоку шифртекста: если r получен из криптографически нестойкого PRNG, то $mask$ равно CBC-блоку предыдущего шифртекста; если r получен из криптографически стойкого PRNG, то $mask = 0$. Один непредсказуемый iv используется для одной пары «открытый текст — MAC-код». В остальном функция безопасности протокола записи соответствует TLS 1.0.

TLS 1.1 исключил использование экспортных шифронаборов и добавил поддержку AES-128/256, однако оставил единственный режим работы блочных шифров CBC. Для TLS 1.1 обязателен криптонабор TLS-RSA-WITH-3DES-EDE-CBC-SHA. Приняты

меры от атак на паддинг в режиме CBC, поэтому изменён порядок обработки ошибок, согласно которому в случае ошибочного заполнения используется оповещение о некорректном MAC-коде сообщения взамен предупреждения некорректного расшифрования. В остальном оповещения об ошибках совпадают с TLS 1.0.

1.5. Протокол TLS 1.2

Первый цикл TLS 1.2 [10], как и в предыдущих версиях, содержит расширения ext, которые теперь объединены в одну спецификацию [11]. Зачастую они содержат информацию о поддерживаемых парах «хеш-функция — алгоритм подписи». Версия TLS 1.2 имеет значение 0x0303.

В TLS 1.2 для RSA-подписи используется схема RSASSA-PKCS1-v1.5 [7] с одним хешированием вместо пары MD5/SHA1 в ранних версиях. Подписываемое сообщение содержит информацию о применяемом дайджесте в DER-кодировке. Более ранние версии TLS не включали информацию о дайджесте при кодировании сообщений. Шифрование RSA выполняется с помощью схемы RSAES-PKCS1-v1.5, также определённой в [7]. Применяемые алгоритмы могут задаваться в расширении ext.

В TLS 1.2 структурно функция PRF упрощена и использует только одну функцию HMAC. Секрет s не разделяется на части, поэтому суммирование отдельных промежуточных результатов исключается. Теперь PRF задаётся явным образом криптонаборами в фазе согласования, хотя по умолчанию определена функция HMAC-SHA256. Входные параметры (s, L, $seed$ и a_i) аналогичны PRF из TLS ранних версий:

$$\mathrm{PRF}(s, L, seed) = \text{HMAC-H}(s, a_1 \parallel L \parallel seed) \parallel \ldots \parallel \text{HMAC-H}(s, a_i \parallel L \parallel seed). \qquad (5)$$

Функции поточного и блочного шифрования соответствуют TLS 1.1, однако алгоритмы вычисления MAC-кодов сообщений заменяются более стойкими.

В TLS 1.2 добавлена поддержка аутентифицированного шифрования с ассоциированными данными (AEAD, Authenticated Encryption with Associated Data), представленная режимами CCM (Counter with Cipher Block Chaining — Message Authentication Code) и GCM (Galois/Counter Mode) [12, 13]. На вход AEAD-алгоритма поступают ключ записи k_w, одноразовый номер $nonce$, данные приложения $data$ с необходимым дополнением, а также ассоциированные данные ad (Associated Data), состоящие из номера сообщения SN, типа $comp_{\text{type}}$ и версии $comp_{\text{ver}}$ функции компрессии и длины сжатых данных $comp(data)_{\text{len}}$. Как правило, построение одноразовых номеров $nonce$ включает явную и неявную части [14]. Ключ записи MAC-кода в AEAD-режиме не применяется. Логически AEAD-функцию можно представить в следующем виде:

$$ad = (SN \parallel comp_{\text{type}} \parallel comp_{\text{ver}} \parallel comp(data)_{\text{len}}),$$
$$data_{\text{AEAD}} = \text{AEAD}(k_w, nonce, data, ad).$$

Ключевой блок TLS 1.2 вырабатывается с использованием новой функции PRF (5), входные параметры аналогичны предыдущим версиям протокола. В случае AEAD-шифрования явная часть $nonce$ принимает значение iv_c или iv_s из ключевого блока $keyb$. Структурно функция вычисления $keyb$ соответствует функции (3) для TLS 1.0 и TLS 1.1.

Если явно не указан другой тип сертификата сервера, например PGP [15], то он имеет тип X.509v3 [8]. Публичные ключи сертификатов должны соответствовать выбранному алгоритму обмена ключами. В TLS 1.2 сертификат открытого ключа одного алгоритма подписи может подписываться с использованием другого алгоритма подписи, например ключ RSA может подписываться ключом DSA.

Протокол TLS 1.2 начинает поддерживать криптографию на эллиптических кривых [16]. Для таких алгоритмов обмена ключами должно указываться явно, отправляется ли сообщение ke или нет. Если отправляется, то оно содержит требуемые для обмена $prems$ публичные параметры. Сообщения ke структурно аналогичны предыдущим версиям TLS, однако поддерживают новые алгоритмы, представленные криптонаборами TLS 1.2.

Запрос на сертификат клиента, помимо поддерживаемых типов сертификатов и информации о центрах сертификации, теперь включает список пар «хеш-функция — алгоритм подписи», которые сервер в состоянии верифицировать. Если у клиента отсутствуют необходимые сертификаты, то он отправляет пустой список сертификатов, что отличается от ранних TLS.

Состав и длина предварительного секрета аналогичен ранним версиям TLS, т. е. $prems = (v_c \parallel rnd_c)$. Однако теперь, если $prems$ не проходит проверку, то он рандомизируется и для стороннего наблюдателя дальнейшие вычисления происходят, как если бы $prems$ был корректен. Это необходимо для предотвращения атак на отформатированные согласно PKCS#1 сообщения, направленных на выяснение того, является ли предполагаемый противником секрет $prems$ верным [17, 18]. На основании [18] этих уязвимостей можно избежать путём обработки неверно отформатированных сообщений и несоответствий номеров версий неотличимым от правильно отформатированных RSA-сообщений способом за счёт инициализации $prems$ новыми значениями.

Для TLS 1.1 и выше вводятся правила обработки сообщений $prems$ с более жёстким контролем версий. Если после расшифрования RSA-блока заполнение и длина корректны и v_c равна TLS 1.1 и выше, то $prems = (v_c \parallel rnd_c)$. Для версий v_c ниже или равных TLS 1.0 и отключенных проверках v_c, $prems$ принимает значение расшифрованного RSA-блока. При некорректных заполнениях или длинах расшифрованных RSA-блоков формируется новый $prems = (v_c \parallel R)$ или $prems = R$, где R — 46 или 48 случайных байт сервера. Если v_c не соответствует версии из первого цикла квитирования, то R может быть 48 байт.

Эта техника получила название «Ослепление RSA». В любом случае, сервер TLS 1.2 не должен генерировать предупреждение, если обработка RSA-зашифрованного сообщения $prems$ завершается ошибкой или номер версии оказался неожиданным. Вместо этого он должен продолжать рукопожатие на основе случайно сгенерированного секрета $prems$ как в случае, если бы ошибка отсутствовала. Ошибка в $prems$ будет установлена клиентом, когда он создаст недопустимый главный секрет ms.

Сообщения верификации сертификата клиента crt_{verif} для TLS 1.2 также рассчитываются на основе всех предыдущих сообщений рукопожатия с использованием подписей RSA или DSA, однако теперь для RSA могут применяться любые согласованные алгоритмы хеширования, не противоречащие сертификату. Для DSA определена только функция SHA1, хотя ожидаются изменения [19], которые позволят применять как другие хеши, так и связывать отдельные хеши с разными длинами ключей DSA.

Расчёт финальных сообщений отличается от (4). Двойное хеширование заменено на однократное, соответствующее применяемому в HMAC для PRF (5). Иными словами, хеш-функция H в сообщении fin теперь задаётся криптонаборами явно, в то время как в ранних версиях TLS были жёстко предписаны MD5/SHA1. Длина fin устанавливается согласно шифронабору, но если она не задана явно, составляет 12 байт. Функция расчёта fin имеет следующий вид:

$$fin_{c/s} = \text{PRF}(ms, \text{«client/server finished»}, H(hsmsg)).$$

Протокол TLS 1.2 по сравнению с другими версиями исключает из применения алгоритмы IDEA и DES, включает поддержку SHA256 и принципиально новых схем шифрования AEAD [14], представленных режимами CCM и GCM [12, 13]. Кроме того, добавлен функционал криптографии на эллиптических кривых [16], в котором, например, ECDSA теперь может использоваться с дайджестами, отличными от SHA1. Обязательным шифронабором для TLS 1.2 является TLS-RSA-AES-128-CBC-SHA. В TLS 1.2 добавлено критическое предупреждение для случая, в котором клиент получает ответ от сервера с расширением, отсутствовавшим в запросе клиента.

1.6. Протокол TLS 1.3

Механизм согласования версии TLS 1.2 устарел в пользу списка версий в расширении. В TLS 1.3 [20] расширение s_v (Supported Versions) используется клиентом для указания списка поддерживаемых версий, а сервером — для указания используемой версии протокола. Если оно присутствует в начальном цикле, то сервер не использует версию клиента v_c для согласования. Сервер, который согласовывает версии до TLS 1.3, не отправляет такое расширение. При согласовании TLS 1.3 отправляется $v_s = 0\text{x}0303$ и $s_v = 0\text{x}0304$.

Полное рукопожатие сокращено до трёх циклов. Опционально, первый цикл может включать параметры эллиптической кривой или конечного поля для обмена ключами $keys$ (Key Share), алгоритмы подписи sa (Signature Algorithms), предварительно распределённый ключ PSK (Pre-Shared Key) и режим обмена общими ключами $pskkem$ (PSK Key Exchange Modes) в виде (EC) DHE, PSK или PSK с (EC) DHE. Второй цикл может использовать защищённые данные приложения $data_{\text{app}}$ и обязательно включает зашифрованные расширения ext_{enc} (Encrypted Extensions):

1) $C \to S:\ \ v_c, r_c, sid_c, cs_c, keys^*, sa^*, pskkem^*, PSK^*$;
2) $C \leftarrow S:\ \ v_s, r_s, sid_s, cs_s, keys^*, PSK^*, ext_{\text{enc}}, crt^*_{\text{req}}, crt^*_s, crt^*_{\text{verif}_s}, fin_s, data^*_{\text{app}}$;
3) $C \to S:\ \ crt^*_c, crt^*_{\text{verif}_c}, fin_c$.

Сервер может инициировать перезапрос параметров (EC) DHE для выработки нового общего ключа. Тогда протокол принимает вид

1) $C \leftarrow S:\ \ keys$;
2) $C \to S:\ \ v_c, r_c, sid_c, cs_c, keys$;
3) $C \leftarrow S:\ \ v_s, r_s, sid_s, cs_s, keys, ext_{\text{enc}}, crt^*_{\text{req}}, crt^*_s, crt^*_{\text{verif}_s}, fin_s, data^*_{\text{app}}$;
4) $C \to S:\ \ crt^*_c, crt^*_{\text{verif}_c}, fin_c$.

В процессе квитирования TLS 1.3 выполняет аутентификацию за счёт конкатенации сообщений m_i, $i = 1, 2, \ldots$, на основе хеширования H_T (Transcript-Hash). В перечень m_i в порядке их легитимного поступления входят: $ClientHello$, $HelloRetryRequest$, новый $ClientHello$, $ServerHello$, ext_{enc}, crt_{req_s}, crt_s, crt_{verif_s}, fin_s, ed_{end} (End Early Data), crt_c, crt_{verif_c}, fin_c. В единственном случае — при перезапросе $ClientHello$ — m_1 заменяется специальным синтетическим сообщением. Это сделано для возможности сервера перезапустить квитирование без сохранения состояния, фиксируя только хеш $ClientHello$ в расширении cookie:

$$H_T(m_1, m_2, \ldots, m_i) = H(m_1 \parallel m_2 \parallel \ldots \parallel m_i). \tag{6}$$

Данные cookie также могут использоваться сервером для требования подтверждения доставки сообщений от клиента как защиты от DoS-атак. Для повторного квитирования TLS 1.3 предусматривает отправку фиксированного значения от SHA256

во втором цикле квитирования. Для защиты от атак на понижение версии 8 младших байт r_s устанавливаются в одно из двух специальных значений, определённых отдельно для попыток согласования TLS 1.2 или TLS 1.1 и более ранних версий.

В TLS 1.3 все сообщения рукопожатия после приветствия сервера второго цикла, а также расширения ext_{enc} зашифрованы. Для верификации сертификата crt_{verif} теперь подписываются 64 байта 0x20 для предотвращения атак с выбранным значением r_c на подписи ранних TLS, строка-константа отправителя сообщения, разделительный байт 0x00 и дайджест от функции (6):

$$sig_{s/c} = (\text{0x20 } \ldots \text{ 0x20} \,\|\, \text{«TLS 1.3, server/client CertificateVerify»} \,\| \\ \| \text{ 0x00} \,\|\, H_T(hsmsg, crt_{s/c})).$$

Выработка всех ключей основана на базовом примитиве HKDF. Вычисление финишного сообщения TLS 1.3 производится за счёт HMAC и HKDF [21] на основе базового ключа k_b, константы, длины хеша H_{len}, сообщений протокола рукопожатия, а также сертификата crt и его верификации crt_{verif} (при их наличии), как определено в следующих функциях:

$$k_f = \text{HKDF}(k_b, \text{«finished»}, \text{«»}, H_{\text{len}}),$$
$$fin = \text{HMAC-H}(k_f, H_T(hsmsg, crt, crt_{\text{verif}})).$$

Добавлена поддержка передачи сообщений рукопожатия после выполнения главного рукопожатия, зашифрованных на ключах и алгоритмах трафика приложения. Так, сообщения *newst* (New Session Ticket) создают уникальную связь между значениями билетов и ключами PSK, полученными из возобновлённого главного секрета ms_{res}. Шифронабор восстанавливаемой сессии должен включать HKDF [21] исходной сессии. Ключи PSK могут использоваться для будущих рукопожатий и открытия нескольких параллельных соединений. Состав билета *newst*: 32-разрядное время жизни *time* в секундах (но не более 7 суток), 32-разрядное случайное значение *age* для скрытия истинного возраста билета (суммируется клиентом с *time* $(\text{mod } 2^{32})$ и включается в PSK), одноразовый номер *nonce*, идентификатор *ticket* для ключа PSK и расширение (только данные 0-RTT). Для каждого билета ставится в соответствие уникальный ключ PSK, вычисленный применением функции с дополнительной меткой HKDF_L из [21]:

$$newst = (time \,\|\, age \,\|\, nonce \,\|\, ticket \,\|\, ext),$$
$$PSK = \text{HKDF}_L(ms_{\text{res}}, \text{«resumption»}, nonce, H_{\text{len}}).$$

Вводится опция повышения производительности под названием «нулевое время возобновления приёма-передачи», или 0-RTT (Zero Round Trip Time Resumption), предназначенная для передачи защищённых ранних данных до окончания квитирования. В этом случае клиент использует ранние данные *ed* (Early Data) и расширение для *PSK*, необходимое при согласовании идентификатора предварительно установленного общего ключа, который будет использоваться с данным рукопожатием как указатель на ключ.

Расширение *PSK* состоит из метки ключа, возраста ключа, списка идентификаторов ключей клиента и списка взаимосвязанных пар «HMAC — ключ *PSK*» (Binders), где HMAC связывает *PSK* с конкретным рукопожатием. Каждый *PSK* связан с одним алгоритмом хеширования. Для *PSK*, установленного через билетный механизм,

выбирается HKDF соединения. Для *PSK*, установленного внешним способом, алгоритм хеширования устанавливается одновременно с *PSK* или по умолчанию SHA256. Версия TLS 1.3 фиксирует KDF для связки с *PSK*, однако TLS 1.2 применяет PRF. Протокол квитирования для создания общего ключа *PSK* принимает следующий вид:

1) $C \to S:\ \ v_c, r_c, sid_c, cs_c, keys;$
2) $C \leftarrow S:\ \ v_s, r_s, sid_s, cs_s, keys, ext_{\text{enc}}, crt^*_{\text{req}}, crt^*_s, crt^*_{\text{verif}_s}, fin_s, data^*_{\text{app}};$
3) $C \to S:\ \ crt^*_c, crt^*_{\text{verif}_c}, fin_c;$
4) $C \leftarrow S:\ \ newst.$

Для возобновления сессии на основе выработанного общего *PSK* выполняются три цикла вида

1) $C \to S:\ \ v_c, r_c, sid_c, cs_c, keys^*, PSK;$
2) $C \leftarrow S:\ \ v_s, r_s, sid_s, cs_s, PSK, keys^*, ext_{\text{enc}}, fin_s, data^*_{\text{app}};$
3) $C \to S:\ \ fin_c.$

Для передачи ранних данных до завершения квитирования выполняются следующие циклы:

1) $C \to S:\ \ v_c, r_c, sid_c, cs_c, ed, keys^*, pskkem, PSK, data^*_{\text{app}};$
2) $C \leftarrow S:\ \ v_s, r_s, sid_s, cs_s, PSK, keys^*, ext_{\text{enc}}, ed^*, fin_s, data^*_{\text{app}};$
3) $C \to S:\ \ ed_{\text{end}}, fin_c.$

Криптографические параметры данных 0-RTT поступают совместно с используемым ключом *PSK*. Если ключ *PSK* установлен с помощью билета новой сессии *newst*, то криптографические параметры поступают из согласованного соединения, которое установило *PSK*. Все данные сообщений первого цикла, исключая текущий список пар «HMAC — ключ *PSK*», обрабатываются аналогично финальному сообщению, но с ключом *PSK*. Сообщение fin_s является последним сообщением, зашифрованным с помощью ключа *PSK* для 0-RTT, после него ключ заменяется.

Примечательно, что в протоколе TLS 1.3 нет встроенной защиты от атак воспроизведения для 0-RTT, поэтому необходим дополнительный механизм ограничения количества повторов. Спецификация TLS 1.3 предлагает комбинацию из трёх контрмер. Первая — это одноразовые билеты или их использование в качестве ссылок на базы данных с *PSK*. Вторая — запись уникальных значений r_c и связующих HMAC для *PSK* в пределах заданного временного окна для выявления дубликатов. Третья — это контроль свежести $ClientHello$ на основе времени отправки сообщений. Сервер может включать в билет время его создания, суммированное с допустимым временем жизни. Клиент вычисляет возраст билета как разность между ticket-age-add из билета и obfuscated-ticket-age из *PSK*. В общем случае следует отправлять только такие данные 0-RTT, которые безопасны для воспроизведения.

Новая концепция шифронаборов TLS 1.3 отделяет механизмы аутентификации и обмена ключами от алгоритмов защиты записи (включая длину секретного ключа), одна хэш-функция используется для вывода ключа и MAC-кода сообщений. Удалено сжатие данных. Для защиты данных используются исключительно AEAD-режимы. Ассоциированные данные состоят из типа записи $content_{\text{type}}$ (шифронабор, рукопожатие, данные приложения и т. д.), устаревшей версии записи (0x0303 соответствует

TLS 1.2) и длины блока зашифрованного текста. Данные приложения *data* содержат тип контента, версию протокола, длину и фрагмент данных. В блок открытого текста *pt* входят данные *data*, тип контента и заполнение нулевыми байтами zeros. Для расшифрования в функцию AEAD вместо *pt* подаются $data_{\text{AEAD}}$. Одноразовое значение *nonce* (не менее 8 байт) строится на основе 64-разрядного порядкового номера SN, дополненного нулевыми байтами в старших разрядах до длины вектора инициализации, которое суммируется (mod 2) с ключом записи k_w. В отличие от TLS 1.2, *nonce* для TLS 1.3 не разделяется на части и полностью вычисляется явно:

$$ad = (content_{\text{type}} \parallel v \parallel ct_{\text{len}}),$$
$$data = (content_{\text{type}} \parallel v \parallel \text{len} \parallel \text{fragment}),$$
$$pt = (data \parallel content_{\text{type}} \parallel \text{zeros}),$$
$$data_{\text{AEAD}} = \text{AEAD}(k_w, nonce, ad, pt).$$

Выработка ключей TLS 1.3 происходит за счёт функций расширения HKDF_{EXP}, экстрагирования HKDF_{EXT} и расширения с дополнительной меткой HKDF_L [21], в которых используются хеш-функции, согласованные криптонаборами. Входными источниками энтропии являются секреты *PSK* и общий секрет (EC) DHE. По сравнению с ранними версиями, TLS 1.3 заменяет PRF на HKDF и использует более сложный механизм расчёта для большего количества ключей. Также определена дополнительная функция выработки секрета DS (Derive Secret):

$$\text{HKDF}_L(s, L, context, \text{len}) = \text{HKDF}_{\text{EXP}}(s, \text{len} \parallel \text{«tls13»} \parallel L \parallel context, \text{len}),$$
$$\text{DS}(s, L, hsmsg) = \text{HKDF}_L(s, L, H_T(hsmsg), H_{\text{len}}).$$

Все основополагающие ключевые материалы TLS 1.3 взаимосвязаны криптографически и последовательно вытекают друг из друга путём вычисления в определённом порядке. В первую очередь вычисляется ранний секрет s_{early}, затем ключ функции HMAC для идентификации *PSK* k_{binder}, ключ клиента данных 0-RTT/экспортер главного секрета 0-RTT $k_{\text{early}}/ms^e_{\text{early}}$, промежуточное значение ds_1, главный секрет рукопожатия s_{HS}, секрет трафика рукопожатия клиента/сервера $s_{\text{HS}_{c/s}}$, промежуточное значение ds_2, главный секрет ms, секрет трафика приложения клиента/сервера $s_{\text{app}_{c/s}}$ и, наконец, экспортер главного секрета/возобновляемый главный секрет $ms_{\text{exp/res}}$. В спецификации зафиксированы значения const. Значение «0» обозначает строку из H_{len} байтов 0x00. Примечательно, что если *PSK* не используется, то ранний секрет s_{early} вычисляется из двух нулевых строк:

$$s_{\text{early}} = \text{HKDF}_{\text{EXT}}(\text{«0»}, PSK);$$
$$k_{\text{binder}} = \text{DS}(s_{\text{early}}, \text{const}_1, \text{«»});$$
$$k_{\text{early}}/ms^e_{\text{early}} = \text{DS}(s_{\text{early}}, \text{const}_{2/3}, ClientHello); \quad (7)$$
$$ds_1 = \text{DS}(s_{\text{early}}, \text{const}_4, \text{«»});$$
$$s_{\text{HS}} = \text{HKDF}_{\text{EXT}}(ds_1, \text{(EC) DHE}); \quad (8)$$
$$s_{\text{HS}_{c/s}} = \text{DS}(s_{\text{HS}}, \text{const}_{5/6}, ClientHello \ \ldots \ ServerHello); \quad (9)$$
$$ds_2 = \text{DS}(s_{\text{HS}}, \text{const}_7, \text{«»});$$
$$ms = \text{HKDF}_{\text{EXT}}(ds_2, \text{«0»});$$
$$s_{\text{app}_{c/s}} = \text{DS}(ms, \text{const}_{8/9}, ClientHello \ \ldots \ fin_s); \quad (10)$$
$$ms_{\text{exp/res}} = \text{DS}(ms, \text{const}_{10/11}, ClientHello \ \ldots \ fin_{s/c}). \quad (11)$$

Статические наборы шифров RSA и DH удалены из TLS 1.3. Режим обмена общими ключами определяется клиентом. Если он предлагает *PSK*, сервер отправляет новый билет сессии из предложенных режимов. При форматировании согласованный ключ DHE в (8) дополняется ведущими нулями до требуемого размера, что отличается от предыдущих версий TLS. Для кривых secp256r1, secp384r1 и secp521r1 вычисление выполняется согласно [22] по схеме ECKAS-DH1, выработанный секрет представлен X-координатой общей точки эллиптической кривой в виде строки октетов. Для кривых x25519 и x448 работа основана на [23], выход используется без дополнительной коррекции.

Итоговые ключи и векторы инициализации генерируются на основе переменных предыдущих вычислений, в которых для 0-RTT данных значение s соответствует (7), для квитирования (9), для данных приложения (10). Обновление секрета трафика после окончания квитирования инициируется соответствующим запросом (Key Update) и основывается на энтропии предыдущих секретов:

$$k_w = \mathrm{HKDF}_L(s, \text{«key»}, \text{«»}, k_{\text{len}}),$$
$$iv_w = \mathrm{HKDF}_L(s, \text{«iv»}, \text{«»}, iv_{\text{len}}),$$
$$s_{\text{app}(n+1)} = \mathrm{HKDF}_L(s_{\text{app}(n)}, \text{«traffic upd»}, \text{«»}, H_{\text{len}}).$$

Экспортируемый ключевой материал генерируется на основе секрета s, принятого из (7) или (11). Отсутствие и пустое значение *context* дают эквивалентные результаты, что отличает TLS 1.3 от предыдущих версий, в которых результаты не совпадали:

$$\mathrm{TLS}_{\text{EXP}}(L, context, k_{\text{len}}) = \mathrm{HKDF}_L(DS(s, L, \text{«»}), \text{«exporter»}, H(context), k_{\text{len}}).$$

Версия TLS 1.3 включает пять шифронаборов, первые три обязательны: TLS-AES-128-GCM-SHA256, TLS-AES-256-GCM-SHA384, TLS-CHACHA20-POLY1305-SHA256, TLS-AES-128-CCM-SHA256, TLS-AES-128-CCM-8-SHA256. К обязательным схемам цифровых подписей для сертификатов относятся RSA-PKCS1-SHA256, для сертификатов и их верификации RSA-PSS-RSAE-SHA256, а также ECDSA-secp256r1-SHA256. Обязательные эллиптические кривые для обмена ключами secp256r1, x25519 [23].

Зафиксирована поддержка дополнительных схем подписи RSASSA-PKCS1-v1.5, ECDSA на основе кривых secp384r1 и secp521r1, RSASSA-PSS (RSAE и PSS) совместно с SHA256, SHA384 и SHA512, а также EdDSA на основе эллиптических кривых Эдвардса ed25519 и ed448. Схемы RSA-PKCS1 и ECDSA совместно с SHA1 применяются исключительно в целях обратной совместимости. Протокол TLS 1.3 подписывает параметры используемых эллиптических кривых в ECDSA, а также взамен согласования делает выбор в пользу одной точки для одной кривой. Запрещается MD5, SHA224, DSA и сертификаты OpenPGP. Список оповещений о предупреждениях и ошибках расширен до 34 вариантов.

Важно отметить, что на базе прототипа TLS 1.3 ведётся разработка протокола RuTLS [24]. Модификация касается изменений логики взаимодействия участников протокола в различных режимах, используемых криптографических примитивов и шифронаборов, а также системы выработки ключей за счёт поддержки криптографических схем Лимонник-3 [25], Крокус [26] и Эхинацея-2(3) [27], разработанных в Российской Федерации.

В связи с нацеленностью RuTLS на повышение безопасности соединений исключается режим 0-RTT. Данные приложений передаются только после завершения процесса аутентификации, использование RawKeys исключено. Генерация секретных и открытых случайных значений использует только разные генераторы случайных чисел.

Новая ключевая система протокола позволяет формировать различные ключи шифрования/имитозащиты с использованием исключительно групп точек эллиптической кривой, согласование общей точки EC DH обязательно. Генерация ключей применяет идентификаторы участников протокола, извлекаемые из сертификатов согласно [27].

2. Развитие криптографического протокола IPsec: IKE, AH, ESP

2.1. Протокол IKEv1

Спецификация IKEv1 [28] представляет собой гибридный протокол, использующий функционал протоколов Oakley [29], SKEME [30] и ISAKMP [31], предназначенный для согласования ассоциации безопасности SA (Security Association) и выработки аутентифицированного ключевого материала в защищённом виде для последующего использования в AH/ESP.

Фаза 1 протокола IKEv1 предназначена для создания безопасного аутентифицированного канала связи ISAKMP SA, посредством которого происходит защищённый информационный обмен для фазы 2, отвечающей за согласование криптографических параметров различных сервисов, таких, как IPsec. Фаза 1 характеризуется криптографическими параметрами только для канала ISAKMP (IKEv1). Следующие атрибуты используются IKEv1 и согласовываются как часть ISAKMP SA: алгоритм шифрования, алгоритм хеширования для PRF/HMAC, метод проверки подлинности, группа DH.

Основной режим (Main Mode) включает два сообщения согласования политики обмена, два сообщения ценностей DH и *nonce* (n), а также два сообщения аутентификации параметров DH. Агрессивный режим (Aggressive Mode) включает два сообщения согласования политики обмена (аутентификация ответчика во втором сообщении), два сообщения ценностей DH (третье сообщение подтверждает подлинность инициатора и его участие в обмене) и вспомогательных данных DH, возведение в степень допустимо отложить до переговоров.

Main Mode и Aggressive Mode разрешают четыре метода аутентификации: цифровую подпись, две формы аутентификации с асимметричным шифрованием и PSK. Секретное значение k_s вычисляется отдельно для каждого метода аутентификации на основе *nonce* инициатора и респондента (n_I, n_R), общего секрета DH (g^{xy}), cookie (cookie_I, cookie_R) и PSK. Ключи k_s, определенные в порядке для случаев применения в подписи, асимметричном шифровании и PSK, представлены ниже. Функция H обозначает согласованную хеш-функцию:

$$\begin{gathered} k_s = \text{PRF}(n_I \parallel n_R, g^{xy}), \\ k_s = \text{PRF}(H(n_I \parallel n_R), \text{cookie}_I \parallel \text{cookie}_R), \\ k_s = \text{PRF}(PSK, n_I \parallel n_R). \end{gathered} \tag{12}$$

В результате работы Main Mode и Aggressive Mode вырабатываются три группы взаимосвязанного аутентифицированного ключевого материала: k_{s_d} используется для выработки ключей, например для IPsec, k_{s_a} применяется в ISAKMP SA для аутентификации сообщений, k_{s_e} — в ISAKMP SA для конфиденциальности сообщений. Сразу после выработки указанных ключей информационный обмен IKEv1 становится криптографически защищённым:

$$\begin{gathered} k_{s_d} = \text{PRF}(k_s, g^{xy} \parallel \text{cookie}_I \parallel \text{cookie}_R \parallel \text{0x00}), \\ k_{s_a} = \text{PRF}(k_s, k_{s_d} \parallel g^{xy} \parallel \text{cookie}_I \parallel \text{cookie}_R \parallel \text{0x01}), \\ k_{s_e} = \text{PRF}(k_s, k_{s_a} \parallel g^{xy} \parallel \text{cookie}_I \parallel \text{cookie}_R \parallel \text{0x02}). \end{gathered}$$

Аутентификация основывается на цифровых подписях, асимметричных алгоритмах и *PSK* за счёт значений H_I и H_R, полученных как дайджесты от всей полезной нагрузки, включая тип идентификатора, порт и протокол, но исключая общий заголовок. Применяется всё тело полезной нагрузки *SA*, исключая общий заголовок ISAKMP, а также полезная нагрузка для идентификации (id_I, id_R):

$$H_I = \mathrm{PRF}(k_s, g^{x_I} \parallel g^{x_R} \parallel \mathrm{cookie}_I \parallel \mathrm{cookie}_R \parallel SA_I \parallel id_I); \tag{13}$$

$$H_R = \mathrm{PRF}(k_s, g^{x_R} \parallel g^{x_I} \parallel \mathrm{cookie}_R \parallel \mathrm{cookie}_I \parallel SA_I \parallel id_R). \tag{14}$$

В фазе 1 аутентификация на основе подписей *sig* с использованием сертификатов открытого ключа *crt* происходит над значениями H_I (13) и H_R (14), при этом Main Mode включает шесть циклов. Если алгоритм подписи привязан к определённому алгоритму хэширования (например, DSS использует SHA с 160-битным выходом), то используется версия HMAC зафиксированного алгоритма хэширования. Согласованные PRF и дайджест используются для всех других псевдослучайных функций. Как и прежде, символ «*» сигнализирует об опциональном параметре, *hdr* обозначает заголовки ISAKMP согласно [31]:

$$\begin{aligned}
&1) \quad I \to R: \quad hdr_I, SA_I;\\
&2) \quad I \leftarrow R: \quad hdr_R, SA_R;\\
&3) \quad I \to R: \quad hdr_I, ke_I, n_I;\\
&4) \quad I \leftarrow R: \quad hdr_R, ke_R, n_R;\\
&5) \quad I \to R: \quad E_{k_{s_e}}(hdr_I, id_I, crt_I^*, sig_I(H_I));\\
&6) \quad I \leftarrow R: \quad E_{k_{s_e}}(hdr_R, id_R, crt_R^*, sig_R(H_R)).
\end{aligned}$$

Для аутентификации на основе шифрования Main Mode применяет одноразовые номера n и идентификаторы id, защищённые на открытом ключе получателя e. В этом случае циклы 3–6 заменяются другими циклами рукопожатия. Аутентификация на основе *PSK* заменяет циклы 5 и 6 на циклы, содержащие id_I (H_I) и id_R (H_R), в которых k_{s_e} является производным от *PSK* из (12):

$$\begin{aligned}
&3') \quad I \to R: \quad hdr_I, ke_I, H(crt_R), E_{e_R}(id_I), E_{e_R}(n_I);\\
&4') \quad I \leftarrow R: \quad hdr_R, ke_R, E_{e_I}(id_R), E_{e_I}(n_R);\\
&5') \quad I \to R: \quad E_{k_{se}}(hdr_I), H_I;\\
&6') \quad I \leftarrow R: \quad E_{k_{se}}(hdr_R), H_R.
\end{aligned}$$

При аутентификации на основе изменённого режима шифрования *nonce* зашифровывается с использованием открытого ключа получателя e. Однако обмен DH (ke), id и crt зашифровываются с использованием согласованного симметричного алгоритма из *SA* на симметричном ключе k, выведенном из *nonce*. Это решение экономит две дорогостоящих операции с открытым ключом на каждой стороне. Здесь циклы $3'$ и $4'$ заменяются:

$$\begin{aligned}
&3'') \quad I \to R: \quad hdr_I, H(crt_R), E_{e_R}(n_I), E_{k_I}(ke_I), E_{k_I}(id_I), E_{k_I}(crt_I);\\
&4'') \quad I \leftarrow R: \quad hdr_R, E_{e_I}(n_R), E_{k_R}(ke_R), E_{k_R}(id_R).
\end{aligned}$$

В Aggressive Mode аутентификация на основе подписей представлена тремя циклами, в которых заголовки hdr передаются в открытом виде:

$$\begin{aligned}
&1) \quad I \to R: \quad hdr_I, SA_I, ke_I, n_I, id_I;\\
&2) \quad I \leftarrow R: \quad hdr_R, SA_R, ke_R, n_R, id_R, crt_R^*, sig_R;\\
&3) \quad I \to R: \quad hdr_I, crt_I^*, sig_I.
\end{aligned}$$

Aggressive Mode с аутентификацией на основе шифрования с открытым ключом также состоит из трёх циклов:

$$\begin{aligned}
&1) \quad I \to R: \quad hdr_I, SA_I, H(crt_R), ke_I, E_{e_R}(id_I), E_{e_R}(n_I);\\
&2) \quad I \leftarrow R: \quad hdr_R, SA_R, ke_R, E_{e_I}(id_R), E_{e_I}(n_R), H_R;\\
&3) \quad I \to R: \quad hdr_I, H_I.
\end{aligned}$$

Aggressive Mode с аутентификацией на основе изменённого режима шифрования принимает следующий вид:

$$\begin{aligned}
&1) \quad I \to R: \quad hdr_I, SA_I, H(crt_R), E_{e_R}(n_I), E_{k_I}(ke_I), E_{k_I}(id_I), E_{k_I}(crt_I);\\
&2) \quad I \leftarrow R: \quad hdr_R, SA_R, E_{e_I}(n_R), E_{k_R}(ke_R), E_{k_R}(id_R), H_R;\\
&3) \quad I \to R: \quad hdr_I, H_I.
\end{aligned}$$

При аутентификации с помощью сертификатов и шифрования на открытом ключе отсутствуют доказательства факта информационного обмена между I и R, так как обе стороны могут полностью реконструировать сообщения.

В Aggressive Mode аутентификация на основе PSK представлена значениями H_I и H_R:

$$\begin{aligned}
&1) \quad I \to R: \quad hdr_I, SA_I, ke_I, n_I, id_I;\\
&2) \quad I \leftarrow R: \quad hdr_R, SA_R, ke_R, n_R, id_R, H_R;\\
&3) \quad I \to R: \quad hdr_I, H_I.
\end{aligned}$$

Симметричные ключи шифрования k являются эфемерными и вычисляются из величин n_{E_I} и n_{E_R} развертыванием до необходимой длины. В режиме CBC вектор инициализации iv первого блока равен нулю, для последующих блоков — последнему блоку шифртекста. Паддинг состоит из 0x00 и оканчивается значением количества дополненных байт:

$$\begin{gathered}
n_{E_{I/R}} = \mathrm{PRF}(n_{I/R}, \mathrm{cookie}_{I/R}),\\
k = (k_1 \parallel k_2 \parallel k_3 \parallel \ldots),\\
\text{где}\ \ k_1 = \mathrm{PRF}(n_{E_{I/R}}, 0\mathrm{x}00),\ k_2 = \mathrm{PRF}(n_{E_{I/R}}, k_1),\ k_3 = \mathrm{PRF}(n_{E_{I/R}}, k_2),\ \ldots
\end{gathered}$$

В фазе 2, или Quick Mode, согласуются дочерние SA и передаются одноразовые номера n для защиты от воспроизведения и создания нового ключевого материала, в параметре ke может согласовываться секрет DH. Допускается одновременное согласование нескольких экземпляров SA.

Quick Mode без ke только обновляет ключевой материал по данным фазы 1 и не предоставляет PFS (Perfect Forward Secrecy), в то время как DH обеспечивает PFS. Протокол ниже описывает Quick Mode, где id клиентов используются для направления

трафика в соответствующий туннель, а идентификатор сообщения m_{id} принимается из заголовка ISAKMP. В режиме CBC для первого сообщения Quick Mode вектор iv вырабатывается (согласованной) хеш-функцией от конкатенации последнего блока CBC фазы 1 и идентификатора сообщения m_{id} фазы 2. Последний блок CBC фазы 1 сохраняется в состоянии ISAKMP SA для синхронизации iv при их запросе в Quick Mode и информационном обмене:

$$\begin{aligned}
&1) \quad I \to R: \quad E_k(hdr_I), H_1, SA_I, n_I, ke_I^*, id_I^*, id_R^*;\\
&2) \quad I \leftarrow R: \quad E_k(hdr_R), H_2, SA_R, n_R, ke_R^*, id_I^*, id_R^*;\\
&3) \quad I \to R: \quad E_k(hdr_I), H_3.
\end{aligned}$$

Значения H_1, H_2 и H_3 вычисляются в следующем виде:

$$\begin{gathered}
H_1 = \mathrm{PRF}(k_{s_a}, m_{id_I} \parallel SA_I \parallel n_I \parallel ke_I^* \parallel id_I^* \parallel id_R^*),\\
H_2 = \mathrm{PRF}(k_{s_a}, m_{id_R} \parallel n_I \parallel SA_R \parallel n_R \parallel ke_R^* \parallel id_I^* \parallel id_R^*),\\
H_3 = \mathrm{PRF}(k_{s_a}, 0\mathrm{x}00 \parallel m_{id_I} \parallel n_I \parallel n_R).
\end{gathered}$$

Ключевой материал дочерней SA строится с применением индекса параметров безопасности SPI (Security Parameter Index). Если требуется свойство PFS, то ke (g^{xy}) принимается в расчёт:

$$k_m = \mathrm{PRF}(k_{s_d}, g^{xy*} \parallel Proto \parallel SPI \parallel n_I \parallel n_R).$$

При необходимости ключевой материал может расширяться:

$$k_m = (k_1 \parallel k_2 \parallel k_3 \parallel \ldots),$$

где

$$\begin{gathered}
k_1 = \mathrm{PRF}(k_{s_d}, g^{xy*} \parallel Proto \parallel SPI \parallel n_I \parallel n_R),\\
k_2 = \mathrm{PRF}(k_{s_d}, k_1 \parallel g^{xy*} \parallel Proto \parallel SPI \parallel n_I \parallel n_R),\\
k_3 = \mathrm{PRF}(k_{s_d}, k_2 \parallel g^{xy*} \parallel Proto \parallel SPI \parallel n_I \parallel n_R), \ \ldots
\end{gathered}$$

По умолчанию поддерживается две группы конечных полей и две группы эллиптических кривых. Доступен режим New Group Mode для согласования новых параметров DH, который запускается в конце фазы 1. Доступна передача только идентификатора группы или кривой. Циклы New Group Mode имеют следующий вид:

$$\begin{aligned}
&1) \quad I \to R: \quad E_k(hdr_I), H_1, SA_I;\\
&2) \quad I \leftarrow R: \quad E_k(hdr_R), H_2, SA_R.
\end{aligned}$$

Параметры H_1 и H_2 представляются в виде

$$\begin{aligned}
H_1 &= \mathrm{PRF}(k_{s_a}, m_{id_I} \parallel SA_I),\\
H_2 &= \mathrm{PRF}(k_{s_a}, m_{id_R} \parallel SA_R).
\end{aligned}$$

Реализации IKEv1 должны поддерживать DES-CBC с контролем слабых и полуслабых ключей [32], 3DES, MD5, SHA, Tiger, аутентификацию на основе *PSK* и DSA, подписи и аутентификацию на основе RSA. По умолчанию для DH определены 768- и 1024-разрядные группы (mod p), 155- и 185-разрядные группы эллиптических кривых. Также могут поддерживаться IDEA-CBC, Blowfish-CBC, RC5-R16-B64-CBC, CAST-CBC. Примечательно, что источник [28] не определяет структуру функции PRF.

2.2. Протокол IKEv2

Протокол IKEv2 [33] объединяет ранние документы [28, 31, 34] и поддерживает работу через NAT. В этом случае туннелируемые пакеты инкапсулируются в UDP для идентификации конечных узлов по номеру порта. IKEv2 заменяет восемь различных начальных обменов IKEv1 одним обменом с четырьмя сообщениями. Механизм аутентификации реализован на основе единственного поля $auth$ взамен рапределения частей данных аутентификации по всему обмену IKEv1. В рамках противодействия атакам на понижение, помимо явного указания версий в сообщениях, новая спецификация вводит дополнительный флаг оповещения о поддержке более высоких версий IKE.

В базовом обмене первые два цикла теперь носят название инициализации IKE (IKE INIT). Здесь hdr содержит SPI, номера версий и флаги. В SA указываются криптографические параметры инициатора для IKE SA, ke содержит значения DH. Одноразовые номера n должны быть не короче 128 бит и не короче половины длины ключа PRF. Ответчик выбирает криптонабор SA_{R_1} из списка инициатора SA_{I_1}. Запрос на сертификат crt_{req} включает список дайджестов SHA1 открытых ключей доверенных центров сертификации и представляет собой предложение сертификатов, а не требование их использования. На этом этапе каждая сторона может сгенерировать ключевое зерно $seed_k$ для выработки ключей шифрования величин ke и ключей аутентификации k_a для IKE SA, а также ключ k_d для последующего использования в AH и ESP. Все последующие сообщения, в том числе информационные, защищены согласованными алгоритмами и выработанными ключами.

Третий и четвёртый циклы отвечают за аутентификацию IKE (IKE AUTH). Идентичности сторон подтверждаются значениями id, доказывается знание секретов k_e и k_a, за целостность данных отвечает поле $auth$. Сертификаты crt и список корневых центров сертификации crt_{req} опциональны. При наличии сертификат используется для проверки целостности $auth$. Опция id_R позволяет инициатору указать ответчика переговоров, например когда один IP-адрес содержит несколько идентичностей. Согласование IPsec SA начинается с параметров SA_{I_2}.

Для обмена информацией из SPD (Security Policy Database) введены селекторы трафика ts (Traffic Selector), которые содержат параметры пакетов нового SA: диапазон IP-адресов, диапазон портов, идентификатор протокола IP.

Два заключительных цикла предлагают SA, одноразовые номера, опциональные значения DH и селекторы ts, которые связывают трафик с конкретной SA. Если, например, служба AH или ESP требует новые ключи, отличные от существующих в IKE SA, то в предпоследнем цикле указывается (опциональное) соответствующее информационное сообщение Inf. Также оно может обеспечивать динамическое выделение адресов для удалённого доступа через шлюз IPsec. Селекторы трафика опускаются, если поступил запрос на изменение ключа IKE SA. Все IKE SA идентифицируются по одному полю — SPI:

$$
\begin{aligned}
&1)\quad I \to R: \quad hdr_I, SA_{I_1}, ke_I, n_I;\\
&2)\quad I \leftarrow R: \quad hdr_R, SA_{R_1}, ke_R, n_R, crt^*_{\text{req}};\\
&3)\quad I \to R: \quad hdr_I, E_{k_e,k_a}(id_I, crt^*_I, crt^*_{\text{req}}, id^*_R, auth, SA_{I_2}, ts_I, ts_R);\\
&4)\quad I \leftarrow R: \quad hdr_R, E_{k_e,k_a}(id_R, crt^*_R, auth, SA_{R_2}, ts_I, ts_R);\\
&5)\quad I \to R: \quad hdr_I, E_{k_e,k_a}(Inf^*, SA_I, n_I, ke^*_I, ts^*_I, ts^*_R);\\
&6)\quad I \leftarrow R: \quad hdr_R, E_{k_e,k_a}(SA_R, n_R, ke^*_R, ts^*_I, ts^*_R).
\end{aligned}
$$

В рамках аутентификации инициатор подписывает данные первого цикла, объединённые с n_R и $\mathrm{PRF}(k_{P_I}, id_I)$, а ответчик — данные второго цикла, объединённые с n_I и $\mathrm{PRF}(k_{P_R}, id_R)$ (ключи k_{P_I} и k_{P_R} описаны ниже). Новые группы DH и cookie включаются в подпись (при их наличии). Инициатор и ответчик могут использовать общие ключи или открытые ключи из сертификатов независимо друг от друга в любой комбинации. Для общего ключа аутентификатор $auth$ вычисляется так:

$$auth = \mathrm{PRF}(\mathrm{PRF}(k_{P_{I/R}}, \text{«Key Pad for IKEv2»}), \text{цикл } 1/2).$$

Протокол IKEv2 вводит расширяемые методы аутентификации EAP (Extensible Authentication Protocol) [35]. В этом случае циклы 3–5 базового обмена претерпевают изменения. Протокол EAP обычно используется для аутентификации инициатора совместно с аутентификацией ответчика на основе подписи. Запрос на EAP объявляется исключением поля $auth$ из соответствующего сообщения квитирования. Если респондент желает использовать EAP, он включает соответствующие данные в ответный цикл. Параметры SA_{R_2}, ts_I, ts_R передаются после прохождения аутентификации. Если EAP создает общий ключ как побочный эффект аутентификации, то он используется (только) для генерации поля $auth$ в заключительных циклах, иначе применяются ключи k_{P_I} и k_{P_R}. EAP без выработки общего ключа небезопасны [36].

В EAP определены типы для идентификаторов, уведомлений, в том числе в случае отсутствия поддержки запрашиваемого метода аутентификации, MD5-вызовов, одноразовых паролей (OTP) и общей токен-карты (GTC). Успешная аутентификация EAP индуцируется как EAP*:

$$\begin{aligned}
&3') \quad I \to R: \quad hdr_I, E_{k_e,k_a}(id_I, crt^*_{\mathrm{req}}, id^*_R, SA_{I_2}, ts_I, ts_R);\\
&4') \quad I \leftarrow R: \quad hdr_R, E_{k_e,k_a}(id_R, crt^*_R, auth, \mathrm{EAP});\\
&5') \quad I \to R: \quad hdr_I, E_{k_e,k_a}(\mathrm{EAP});\\
&6') \quad I \leftarrow R: \quad hdr_R, E_{k_e,k_a}(\mathrm{EAP}^*);\\
&7') \quad I \to R: \quad hdr_I, E_{k_e,k_a}(auth);\\
&8') \quad I \leftarrow R: \quad hdr_R, E_{k_e,k_a}(auth, SA_{R_2}, ts_I, ts_R).
\end{aligned}$$

Информационные обмены ведутся в защищённом виде на основе установленных *SA* для IKE или AH/ESP. При закрытии IKE *SA* все дочерние *SA* для AH/ESP автоматически закрываются. При закрытии AH/ESP *SA* удаляются только они. Для удаления *SA* отправляется запрос с индексом *SPI*.

Введены таймеры повторной передачи, при которых инициатор запоминает запрос до получения ответа. Для увеличения пропускной способности определена опциональная поддержка параллельной обработки нескольких ожидающих сообщений не в хронологическом порядке следования, но в пределах выделенного окна. Ответчик запоминает ответ до получения нового запроса с номером, большим, чем в ответе, просуммированным с размером окна порядковых номеров. Для сопоставления запросов / ответов и определения повторных сообщений формат IKEv2 предлагает 32-разрядные идентификаторы как часть заголовков hdr, равные 0 для первых запросов каждого направления (по два счётчика на каждую сторону). Смена ключей IKE *SA* сбрасывает порядковый номер.

Компрессия [37] настраивается как часть IPsec *SA*, однако отдельно от согласования криптографических параметров. Ассоциации сжатия уничтожаются совместно с AH/ESP *SA*.

В рамках снижения эффективности DoS-атак ответчик может минимизировать использование CPU и исключить хранение SA до подтверждения IP-адреса отправителя. Для этого cookie специального вида включаются в обмен новой инициализации IKE. Однако долгое хранение cookie снижает потенциал защиты от DoS-атак:

$$\text{cookie} = (ver_s \parallel H(n_I \parallel \text{IP}_I \parallel SPI_I \parallel secret)).$$

Включение SPI_I гарантирует разные cookie для разных SA, n_I гарантирует невозможность создания cookie без знания первого цикла базового обмена, $secret$ известен только респонденту, а версия ver_s изменяется при обновлении секрета. Ответ от инициатора должен содержать аналогичную нагрузку. Для описанного случая между циклами 1 и 2 базового обмена добавляются следующие обмены:

$$\begin{aligned} 1'') \quad & I \leftarrow R: \quad hdr_R, \text{cookie}; \\ 2'') \quad & I \rightarrow R: \quad hdr_I, \text{cookie}, SA_{I_1}, ke_I, n_I. \end{aligned}$$

Респондент сравнивает отправленный и принятый cookie. Совпадение означает, что данные созданы после последнего изменения секрета, что гарантирует их актуальность. Кроме того, IP_I должен совпадать с исходным адресом, с которым соединялся респондент.

В IKEv1 время жизни SA согласовывалось, в IKEv2 каждая сторона ведёт собственную политику времени жизни и смены ключей SA. IKEv2 разрешает параллельные SA с одинаковыми селекторами трафика для поддержки различий качества обслуживания (QoS). В отличие от IKEv1, комбинация конечных точек и селекторов трафика однозначно не идентифицирует SA.

Протокол IKEv2 согласует алгоритмы шифрования, целостности, группы DH и PRF, причём PRF используется для вывода ключевого материала как в IKE SA, так и в IPsec SA. В случае длинных ключей применяется итеративное вычисление:

$$\text{PRF}'(k, s) = (t_1 \parallel t_2 \parallel t_3 \parallel \ldots),$$
$$\text{где} \qquad t_1 = \text{PRF}(k, s \parallel \text{0x01}),\ t_2 = \text{PRF}(k, t_1 \parallel s \parallel \text{0x02}),\ t_3 = \text{PRF}(k, t_2 \parallel s \parallel \text{0x03}), \ldots$$

На основе величины $seed_k$ вычисляются семь секретных ключей: k_d — материал для вывода новых ключей в IPsec SA; k_{a_I} и k_{a_R} — ключи целостности алгоритма аутентификации последующих обменов; k_{e_I} и k_{e_R} — ключи симметричного шифрования; k_{P_I} и k_{P_R} — ключи для вычисления поля $auth$. Изменение ключей изменяет SPI. Для нового SA материал $seed'_k$ вычисляется с использованием существующего k_d, обновлённых $nonce$ и, опционально, новых секретов DH. Итеративная функция вырабатывает ключи в определенном порядке (используется $seed_k$ или $seed'_k$). Если PRF принимает ключи фиксированной длины, то конечные биты n_I и n_R поровну отбрасываются:

$$seed_k = \text{PRF}(n_I \parallel n_R, g^{xy}); \tag{15}$$
$$seed'_k = \text{PRF}(k_d, g^{xy*} \parallel n_I \parallel n_R); \tag{16}$$
$$(k_d \parallel k_{a_I} \parallel k_{a_R} \parallel k_{e_I} \parallel k_{e_R} \parallel k_{P_I} \parallel k_{P_R}) = \text{PRF}'(seed_k, n_I \parallel n_R \parallel SPI_I \parallel SPI_R). \tag{17}$$

В IKEv2 добавлена поддержка работы через NAT посредством инкапсуляции данных в пакеты UDP для маршрутизации и идентификации конечных узлов по номеру порта. Соответствующие уведомления включаются после n_I и n_R в базовых циклах (1) и (2). Предотвращение специальной или некорректной обработки IPsec в узлах NAT

реализуется перенаправлением трафика через порты 500/4500. Для идентификации вложенных протоколов в случае IKEv2 после заголовка UDP добавляются 4 байта 0x00, в AH/ESP здесь находится *SPI*.

Проверка факта нахождения NAT между узлами (или какой из узлов находится за NAT) осуществляется на основе двух дайджестов SHA1, вычисленных отдельно от *SPI*, IP-адреса и порта инициатора и респондента. Получатель таких уведомлений сравнивает принятый и собственноручно вычисленный дайджесты SHA1. Если они не совпадают, то собеседник находится за NAT (на маршруте произведено изменение адреса источника исходного пакета для соответствия адресу узла NAT) и следует включить обход NAT. В этом случае система должна позволять динамическое обновление адресов и отправку поддерживающих соединение пакетов согласно [38]. Если дайджесты не соответствуют адресам заголовков, то необходимо туннелировать будущие пакеты, связанные с этим IKE *SA*, через UDP порт 4500.

В транспортном режиме изменение IP-адреса вызывает сбой MAC. Узел NAT не в состоянии откорректировать MAC, так как он криптографически защищён. Однако для будущей проверки MAC отправитель вводит IP-адрес в селекторы трафика. В туннельном режиме возникают проблемы с маршрутизацией, где трансляция адресов AH/ESP требует специальной логики в NAT, которая не является тривиальной. Решение также заключается в инкапсуляции трафика IPsec в UDP.

Для определения динамического IP-адреса узла, расположенного за защищённым шлюзом NAT, используется специальный параметр удалённого доступа IPsec, который включается в циклы 3 и 4 основного рукопожатия перед SA_I и SA_R. Запрос нового адреса может быть включен в любой запрос на создание IPsec *SA*. Если, например, NAT перезапускается, то хосты соединяются с IP-адресом и портом последнего аутентифицированного пакета, инкапсулированного в UDP. Такие пакеты используются для динамического (обнаружения) изменения IP-адреса и порта.

Изменение сетевых идентификаторов выявляется проверкой MAC, для которого изменён адрес и порт, связанный с *SA* аутентифицированного пакета. В таких ситуациях следует динамически обновлять комбинации адреса и порта установленной *SA*. Узел за NAT не должен выполнять такое обновление, если проверенные пакеты имеют разные значения адреса и порта, так как это открывает возможность DoS-атакам. Кроме того, динамическое обновление адреса должно выполняться в ответ на новый пакет, иначе злоумышленник может вернуть старые воспроизведённые адреса.

Туннели IPsec на основе IKEv1 при обработке трафика сбрасывают индикацию явного уведомления о перегрузке ECN (Explicit Congestion Notification) во внешних заголовках IP, что приносит ущерб сети. В IKEv2 реализуется полноценная поддержка ECN за счёт введения функциональности туннелей, согласно [39], и обработки данных, согласно [40].

Криптонаборы для IKEv2 включают алгоритмы шифрования DES-IV64, DES, 3DES, RC5, IDEA, CAST, Blowfish, 3IDEA, DES-IV32, AES-CBC, AES-CTR и без конфиденциальности. Функции PRF представлены HMAC-MD5, HMAC-SHA1, HMAC-Tiger, AES-128-XCBC. Целостность обеспечивается HMAC-MD5-96, HMAC-SHA1-96, DES-MAC, KPDK-MD5, AES-XCBC-96. Группы DH представлены 768- и 1024-разрядными конечными полями. Сжатие обеспечено алгоритмами DEFLATE, LZS и LZJH.

2.3. Протокол IKEv3

Спецификация IKEv3 [41] заменяет и обновляет [33], а также включает разъяснения [42]. Взамен опциональности, селекторы трафика становятся обязательными для

всех обменов. В связи со сложностью реализации исключена поддержка установки периода использования внутреннего IP-адреса. Удален атрибут конфигурации адреса сервера имен NetBios. Допускается обработка сообщений с произвольным порядком инкапсуляции полезной нагрузки в отличие от отклонения в предыдущей версии. Выработка ключевого материала *SA* теперь требует обязательного включения секрета DH.

Повторная передача сообщений IKE *SA* возможна в трёх случаях: для полуоткрытого соединения респондент дублирует свой ответ; для запроса новой *SA* — создаёт и отправляет сообщение новой *SA*; для существующей и аутентифицированной *SA* — игнорирует сообщение. В новой спецификации недостаточно использовать только *SPI* и IP-адрес инициатора для определения соединений, открытых в одностороннем порядке, так как два разных респондента за NAT могут выбрать одинаковый *SPI* инициатора. Теперь для поиска IKE *SA* используется всё сообщение, его хеш и полезная нагрузка n_I. В общем случае респондентом целесообразна однократная повторная передача на каждое подобное сообщение.

Разъяснено использование критического флага, рассматриваемого в контексте типа полезной нагрузки, а не её содержимого. IKEv2 [41] добавляет такой флаг к каждому заголовку полезной нагрузки для перспектив гибкости и прямой совместимости. Если критический флаг установлен и тип полезной нагрузки не распознан, то сообщение отклоняется и ответчик сигнализирует о неподдерживаемой полезной нагрузке (в сигнализации критический флаг отсутствует). Если критический флаг не установлен и тип полезной нагрузки также не распознан, то полезная нагрузка игнорируется без дополнительного оповещения.

Если инициатор поддерживает AEAD-режимы, то в *SA* он должен предлагать два варианта криптонаборов: комплекты AEAD-режимов и комплекты обычного шифрования с отдельными алгоритмами целостности. В отличие от IKEv2, согласно [41], при смене ключей IPsec *SA* селекторы трафика и криптонаборы следует оставлять без изменений. Параллельные запросы *SA* удаляются по наименьшему значению *nonce* побайтовым сравнением (не целочисленным).

Если инициатор не замечает одновременную смену ключей и изменяет свою IKE *SA*, а затем получает запрос на смену *SA*, то он оповещает об этом респондента и избыточные *SA* не создаются. Если инициатор замечает одновременную смену ключей и получает запрос на удаление старой IKE *SA*, то он фиксирует, что респондент не обнаружил одновременную смену ключей. В этом случае инициатор останавливает попытку смены ключа. Важно отметить, что IKEv3 не имеет отдельного механизма повторной аутентификации и проходит её только путём создания новой *SA*.

Введены требования предпочтительных длин ключей PRF (например, для секретов k_d, k_{P_I} и k_{P_R}). Для PRF на основе HMAC предпочтительный размер ключа равен длине вывода базовой хеш-функции, другие PRF должны явно его указывать.

Протокол IKEv2 [33] допускал обновление ключевого материала без обмена ke_I и ke_R, однако IKEv3 [41] всегда включает разделяемый секрет DH. Функции вычисления ключевого зерна $seed_k/seed'_k$ используют PRF старой IKE *SA* и совпадают с (15) и (16). Для сеансовых ключей в (17) поступают данные нового обмена, в том числе функция PRF новой IKE *SA*.

Источник [41] расширяет список событий, в которых необходимы оповещения об ошибках и соответствующая реакция сторон. Для ошибок за пределами установленных соединений IKE *SA*, например в базовых циклах 1 и 2, аналогичных IKEv2, во избежание участия в DoS-атаках необходимо ограничивать скорость ответов на крипто-

графически незащищенные уведомления. Необходим компромисс между перегрузкой сети и минимальным количеством повторных сообщений для установления безопасного соединения. Подобные уведомления могут быть результатом сбоя узла и требуют инициирования проверки работоспособности соединений на основе соответствующих IP-адресов, портов, *SPI* и идентификаторов сообщений. В таких ситуациях незамедлительная реакции в виде изменений состояния *SA* не допускается.

Ошибки аутентификации IKE *SA*, например недействительный общий секрет, идентификатор, поставщик сертификата, сертификат и т. д., вызывают критические предупреждения. Ошибки после аутентификации IKE *SA*, указывающие на различие состояний сторон, предполагают обновление IKE *SA*. Если проверка MAC проходит, но синтаксис сообщения некорректен, вырабатывается фатальное уведомление, требующее закрытия IKE *SA*.

При обходе NAT спецификация [41] вводит требование поддержки обработки как инкапсулированных, так и не инкапсулированных в UDP пакетов IPsec. Любая сторона решает независимо, использовать инкапсуляцию IPsec в UDP или нет, однако при обнаружении NAT обе стороны должны ее использовать. Инкапсуляция в UDP не должна выполняться на порте 500, только 4500. Реализации должны обрабатывать полученные пакеты IPsec, инкапсулированные в UDP, даже если NAT отсутствует.

В результате перекрытия нескольких обменов может возникать ситуация рассинхронизации состояния *SA*. С ростом размера окна параллельной обработки положение значительно усугубляется, особенно при обработке запросов вне очереди. Поэтому для работы в условиях подобных коллизий при смене ключей и закрытии (дочерних) *SA* разработаны дополнительные уведомления.

Введено уведомление временной невозможности ответа на запрос, например в период операции смены/удаления ключей или закрытия *SA* (избыточные *SA* удаляются на основе минимальных *nonce*). В результате отправитель ожидает завершения временного состояния корреспондента. Если подобные уведомления происходят на протяжении длительного времени, например в течение нескольких минут, то соединение рассинхронизировано и требует закрытия.

Другое уведомление отправляется при получении запроса на смену ключа несуществующей дочерней *SA*, где реакцией получателя является удаление такой *SA*.

Добавлено требование обязательной поддержки HTTP-метода [43] для поиска сертификатов по двум значениям, ссылке URL и дайджесту URL. Другие методы могут использоваться в случае выпуска соответствующих спецификаций.

Сетевая конфигурация IPv6 способами, основанными на аналогичных данных IPv4, не соответствует нормальному функционированию IPv6. При автоконфигурации IPv6 состояние узла не сохраняется, рекламные сообщения маршрутизатора и обнаружение соседей становятся недоступны. Для решения подобных осложнений введён экспериментальный документ [44].

В криптонаборах [41] расширен список параметров DH, в который вошли конечные поля с разрядностью модулей 1536, 2048, 3042, 4096, 6144 и 8192 бит.

2.4. Протокол AHv1

Согласно [45], заголовок аутентификации AHv1 вычисляется на основе симметричных или асимметричных криптографических алгоритмов и занимает положение непосредственно после заголовка IPv4 и перед заголовками протоколов верхнего уровня. В случае IPv6 AH обычно занимает положение после заголовка параметров обра-

ботки маршрутизации (Hop-by-Hop) или списка необходимых промежуточных узлов (Routing) и перед параметрами получателя (Destination Options).

Хеш-функции H для вычисления ICV (Integrity Check Value) заголовка AH принимаются из SA, на которые ссылаются IP-адрес и SPI. Обязательна поддержка ключевого MD5 [46]. Обычные контрольные суммы, например CRC-16, криптографически нестойкие, поэтому запрещены.

Для расчёта ICV принимаются практически все данные сообщения. Модифицируемые при транзите поля, например TTL и CRC в IPv4 или Hop Limit в IPv6, входят в расчёт ICV как нулевые строки для подтверждения факта их использования и длины, но не контроля содержимого. Синтаксис AH в порядке следования включает 8-разрядный заголовок полезной нагрузки NH (Next Header), 8-разрядную длину данных аутентификации ICV_{len} как количество 32-разрядных слов, 16-разрядный резерв R, 32-разрядный SPI и непосредственно данные аутентификации ICV переменной длины, кратной 32-разрядному слову:

$$P_{\text{AHv1}} = (\text{NH} \parallel ICV_{\text{len}} \parallel R \parallel SPI \parallel ICV).$$

Данные ICV принимаются за нулевую строку при расчёте. Все данные $Data$ протокола верхнего уровня в контексте AH рассматриваются как полезная нагрузка и включаются в расчёт ICV полностью. Дополнительные метки, например IPSO, принимаются в расчёт ICV.

2.5. Протокол AHv2

Выпуск AHv2 [47] объявляет AHv1 [45] устаревшим. В [47] заголовок аутентификации претерпел изменения. AH стал заголовком расширений в случае IPv6. Для противодействия атакам воспроизведения отдельно для отправителя и получателя добавлены 32-разрядные беззнаковые инкрементные счетчики SN в позицию между SPI и ICV. Исключение зацикливания достигается инициализацией $SN = 0$ при установлении новой SA ранее чем через каждые 2^{32} сообщений. Номер SN всегда включается в AH, однако его проверка оставлена на усмотрение получателя.

Задокументированы списки изменяемых полей заголовков IP, которые не входят в ICV. Для IPv4 это Type of Service, Flags, Fragment Offset, TTL, CRC, а также экспериментальные поля. Для IPv6 это Class, Flow Label, Hop Limit, а также Hop-by-Hop и Destination в том случае, если установлен бит, указывающий, что опция изменяема во время транзита. Тип и длина опции входят в ICV.

Для обеспечения кратности длины AH 32 (IPv4) или 64 (IPv6) разрядам ICV дополняются. Паддинг располагается сразу за ICV, содержит произвольные значения и входит в расчет ICV как данные верхнего протокола. Алгоритмы аутентификации устанавливаются SA и могут включать ключевые MAC на основе симметричных алгоритмов шифрования, например DES, односторонних хэш-функций или хэш-функций в сочетании с асимметричными алгоритмами подписи. Последний вариант актуален для многоадресных решений. Обязательными объявляются HMAC-MD5-96 и HMAC-SHA1-96.

Логическая структура пакета AHv1 и функция вычисления ICV представлены ниже. Неизменяемые поля заголовка IP, принимающие участие в расчёте ICV, обозначены как IP_{HDR}:

$$P_{\text{AHv2}} = (\text{NH} \parallel ICV_{\text{len}} \parallel R \parallel SPI \parallel SN \parallel ICV); \tag{18}$$

$$ICV = H(\text{IP}_{\text{HDR}} \parallel \text{NH} \parallel ICV_{\text{len}} \parallel R \parallel SPI \parallel SN \parallel ICV \parallel Data). \tag{19}$$

Итоговые изменения относительно [45] направлены на создание полноценной платформы AH с интегрированной службой защиты от воспроизведения на основе порядковых номеров сообщений *SN*, с более стойкими бесключевыми алгоритмами HMAC-MD5-96 и HMAC-SHA1-96 вместо ключевого MD5, а также с фиксацией списка исключённых из *ICV* изменяемых заголовков IPv4/IPv6. Туннельный режим с аутентификацией всех данных инкапсулируемого пакета теперь рассматривается как неотъемлемая часть AH, в ранней версии он только упоминался.

2.6. Протокол AHv3

Спецификация [48] внесла дополнения в AHv2 [47] и объявила его устаревшим. Структура пакета AHv3 соответствует AHv2 (18) и (19). Для корректной обработки коллизий *SPI* между одноадресными и многоадресными архитектурами, в которых сервер ключей или групповой контроллер назначают групповую ассоциацию безопасности, источник [48] вводит алгоритм поиска *SA* в базе данных SAD (Security Association Database) на основе параметров *SPI*, IP-адреса назначения и IP-адреса источника. При отсутствии совпадений производится поиск без IP-адреса источника; затем только по *SPI*; затем по *SPI* и протоколу AH/ESP. Найденная *SA* применяется для обработки AH.

Новое расширение включает 64-разрядный порядковый номер *ESN* (Extended Sequence Number) для высокоскоростных коммуникаций. Он устанавливается по умолчанию, если, например, при согласовании IKE явно не указан 32-разрядный *SN*. Младшие 32 бита *ESN* явно передаются в поле SN, старшие биты сохраняются в памяти, а при расчёте *ICV* помещаются сразу после полезной нагрузки и перед неявным заполнением. В AH не предусмотрена синхронизация *SN/ESN* для многоадресных *SA*, поэтому в таких решениях отсутствует защита от повторов.

Вводятся примеры реализаций *SN*-окна антивоспроизведения на основе *ESN*, механизмов его управления, вычисления неявных старших бит *ESN*, а также процесс восстановления синхронизации *ESN* при потере сообщений. Если получатель не поддерживает антивоспроизведение, он не должен согласовывать *ESN* в *SA*. Это исключает необходимость контроля старших бит *ESN* в контексте *SN*-окна антивоспроизведения и расчёта *ICV*.

В расчёте *ICV* обнуляются новые непредсказуемые поля. Для IPv4 это флаги дополнительного обслуживания на маршрутизаторах (DSCP) и их перегрузки (ECN). Запрещается внесение паддинга больше минимума, требуемого для выравнивания границ по 32/64 битам.

Для AHv3 алгоритмы целостности *ICV* могут включать ключевые MAC, основанные на симметричных алгоритмах, например AES, или на хеш-функциях, например MD5, SHA1, SHA256. Обязательные криптографические алгоритмы собраны в едином ресурсе [49] и включают HMAC-SHA1-96, AES-XCBC-MAC-96 и HMAC-MD5-96.

2.7. Протокол ESPv1

Спецификация [50] содержит вводную информацию и предполагает использование ESPv1 в транспортном и туннельном режимах, в которых способы преобразования данных идейно схожи с AH. ESPv1 располагается после IP-заголовка и состоит из 32-разрядного *SPI* в открытом виде и защищённых данных переменной длины *Data* (TCP, UDP и т. д.). Контроль целостности в ESPv1 источник [50] не описывает. Кроме того, сообщение может включать дополнительную открытую нагрузку, передаваемую вместе с ESP. Получатель идентифицирует *SA* по IP-адресу назначения и *SPI*. В обычных условиях ESP не включает явных меток безопасности, так как *SPI* указывает на

уровень защиты. Однако при развертывании многоуровневой системы безопасности должно поддерживаться применение явных меток, например IPSO:

$$P_{\text{ESPv1}} = (SPI \parallel E_k(Data)). \tag{20}$$

Если криптографические алгоритмы ESPv1 дополнительно не обеспечивают аутентификацию, то для таких целей возможно применение AH. В транспортном режиме AH аутентифицирует нагрузку ESPv1 (20) и заголовок IP. В туннельном режиме AH аутентифицирует все поля инкапсулированного IP-пакета и результирующая ESP с зашифрованными AH и полным внутренним IP-пакетом принимается в качестве полезной нагрузки для внешнего IP-пакета.

Обязательным для поддержки в ESPv1 объявлен DES-CBC, однако без включения расчета MAC это небезопасно. Порядок криптографических преобразований всех версий ESP соответствует Encrypt-and-MAC (E&M).

2.8. Протокол ESPv2

Источник [51] признает устаревшим ESPv1 [50] и дает описание ESPv2 как законченного решения. Состав пакета ESPv2, включая обязательные и опциональные параметры, фиксируется конкретной *SA* на время её действия.

Поля пакета ESPv2 в порядке их следования теперь включают 32-разрядный *SPI*, который в сочетании с IP-адресом назначения и протоколом ESP однозначно идентифицирует *SA*. Далее следует 32-разрядный порядковый номер сообщения *SN* для противодействия атакам воспроизведения, который сбрасывается в 0 при установке новой *SA*. Затем располагается полезная нагрузка *Data* переменной длины, содержащая, при необходимости, вектор инициализации *iv* для алгоритма шифрования. Вектор инициализации указывается явно вместе с длиной и позицией расположения или неявно, может быть открытым или частью зашифрованных данных. В любом случае учитываются требования спецификаций, регламентирующих такое поведение. Заполнение *pad* используется для кратности размеру блока алгоритма шифрования, а также выравнивания окончания зашифрованной нагрузки на 4-байтовой границе для гарантии того, что данные аутентификации *ICV* выровнены корректно или для сокрытия длины полезной нагрузки. Если не определено иное, то по умолчанию для простейшей защиты целостности байты заполнения инициализируются серией целочисленных монотонно возрастающих значений. Затем следуют 8-разрядные длина заполнения pad_{len} и следующий заголовок NH, идентифицирующий тип полезной нагрузки. Завершает пакет ESPv2 значение целостности *ICV*. Поле *ICV* включается только в том случае, если аутентификация определена *SA*:

$$P_{\text{ESPv2}} = (SPI \parallel SN \parallel iv^* \parallel E_k(iv^{**} \parallel Data \parallel pad \parallel pad_{\text{len}} \parallel \text{NH}) \parallel ICV).$$

Значение *ICV* вычисляется от пакета ESPv2, исключая поле *ICV*. В транспортном режиме в первую очередь выполняется обработка ESP, а затем фрагментация. В туннельном режиме ESP может применяться к фрагментам IP:

$$ICV = H(P_{\text{ESPv2}}).$$

ESPv2 может использоваться в транспортном и туннельном режимах. В транспортном режиме ESP вставляется после IP-заголовка и перед протоколами верхнего уровня (TCP, UDP, ICMP и т. д.) или заголовками IPsec, которые вставлены ранее. Конфиденциальность IP-заголовка не обеспечена. В IPv4 заголовок ESP находится после IP-

заголовка. В IPv6 заголовок ESP рассматривается как сквозная полезная нагрузка и находится после расширений Hop-by-Hop, Routing или Fragmentation. В туннельном режиме ESP защищает весь инкапсулированный IP-пакет, включая конечные адреса и заголовки. Положение ESP в туннельном режиме относительно внешнего незащищённого IP-заголовка аналогично ESP в транспортном режиме. В обоих режимах аутентифицируются данные от *SPI* в начале ESP и до NH в конце ESP, где внешний заголовок IP не защищён. Заголовки ESP и AH можно комбинировать разными способами.

Алгоритмы конфиденциальности и аутентификации согласуются в *SA*. Для *ICV* могут применяться ключевые MAC на основе симметричных алгоритмов, например DES, хэш-функций, например MD5 или SHA1. Для многоадресных решений могут поддерживаться алгоритмы подписи. Тем не менее в ESP аутентификация является необязательной.

Защита от атак воспроизведения выполняется по аналогии с AH за счёт скользящего окна контроля принимаемых порядковых номеров. Правильная реализация антивоспроизведения требует своевременного изменения состояния *SN*, поэтому в таких сценариях в целях недопущения зацикливания счётчика ручное распределение ключей не допускается. Взлом *SPI* обнаруживается с помощью аутентификации ESP, однако IP-адрес получателя и тип протокола IPsec (AH/ESP) не защищены от активных воздействий. Корректность значений дополнения и его длины проверяются независимо от поля аутентификации.

В ESPv2 обязательными для поддержки являются DES-CBC с явным вектором инициализации, HMAC-MD5-96, HMAC-SHA1-96, а также отсутствие аутентификации или шифрования.

2.9. Протокол ESPv3

Источник [52] описывает использование ESPv3 для обеспечения конфиденциальности, целостности или их обоих одновременно. Услуга только конфиденциальности объявлена как возможная, но не обязательная. Обеспечение целостности ESPv3 является привлекательной альтернативой AH, так как в этом случае отмечается рост производительности и возможность конвейерной обработки. В любом случае, служба антиповтора может использоваться только при наличии *ICV*.

Введена новая функция TFC (Traffic Flow Confidentiality), способствующая эффективной генерации и утилизации фиктивного трафика. Для идентификации фиктивных пакетов в Next Header зафиксировано значение 59. Такие пакеты содержат все заголовки, характерные ESP, однако полезная нагрузка может формироваться произвольным образом, например как случайная последовательность. Данное нововведение направлено на маскирование отсутствия реального трафика.

Дана расширенная модель данных полезной нагрузки и *ICV* для работы с AEAD-алгоритмами, которые дополнительно вычисляют *ICV*. В этом случае в пакете ESPv3 поле *ICV* опускается. Необходимо выбирать такие AEAD-алгоритмы, которые способны поддерживать только аутентификацию без конфиденциальности для полей *SPI*, *SN* и старших бит *ESN* или репликацию указанных полей в зашифрованную нагрузку для обеспечения их целостности. В любом случае, в рамках антивоспроизведения защищённый *SN* должен соответствовать открытому номеру *SN*. Добавлена опция для дополнительной поддержки 64-разрядного *ESN* в высокоскоростных решениях (десятки, сотни Гбит/с).

Формат пакета ESPv3 претерпел изменения. Поля *SPI* и *SN*, расширенное до *ESN*, предоставляют функционал, полностью аналогичный AH [48], включая механизмы

антивоспроизведения и скользящего *SN*-окна. Контроль *SN* выполняется до проверки целостности и расшифрования, что устраняет затраты на криптографические вычисления в случае повтора. При необходимости вектор iv помещается в начале полезной нагрузки $Data$ в незашифрованном виде. Алгоритм шифрования должен дать чёткую спецификацию о структуре iv, его местоположении и способе обработки. Логическая структура пакета ESPv3 с зашифрованными на сеансовом ключе k данными представлена в следующем виде:

$$P_{\text{ESPv3}} = (SPI \parallel ESN^{*} \parallel iv^{*} \parallel E_k(iv^{**} \parallel Data \parallel \text{TFC}^{*} \parallel pad \parallel pad_{\text{len}} \parallel \text{NH}) \parallel ICV).$$

Значение ICV вычисляется от пакета ESPv3, исключая поле ICV. Если используется расширение *ESN*, то для расчёта ICV старшие 32 разряда порядкового номера помещаются за полем NH:

$$ICV = H(P_{\text{ESPv3}}).$$

Дополнение *pad* ограничено 255 байтами, что недостаточно для скрытия характеристик трафика. Если верхний протокол содержит информацию о длине $Data$, то для решения такой задачи TFC производит вставки после полезной нагрузки. Это верно для туннельного и частично верно для транспортного режима, в зависимости от протокола. Использование TFC согласуется *SA* и отличается от фиктивных пакетов большей вариативностью действий. Остальные поля соответствуют ESPv2.

Перечень криптографических алгоритмов ESPv3 внесён в единую спецификацию [49]. Для конфиденциальности требуется поддержка 3DES-CBC, AES-CBC (128-разрядные ключи), AES-CTR и опция без конфиденциальности, DES-CBC не рекомендуется. Обеспечение целостности требует реализации усеченных дайджестов HMAC-SHA1-96, AES-XCBC-MAC-96, возможна HMAC-MD5-96 и опция без целостности.

Заключение

С опорой на базовые спецификации RFC в работе рассмотрены основные этапы развития криптографических протоколов транспортного и сетевого уровней модели OSI на примерах выпусков SSL версий 2.0 и 3.0 [1, 3], TLS версий от 1.0 до 1.3 [4, 6, 10, 20], а также IPsec версий от 1 до 3, состоящих из IKE [28, 33, 41], AH [45, 47, 48] и ESP [51 – 53] соответственно.

Криптопротокол SSL 2.0 рассмотрен в качестве исторической справки как опорная точка первого представителя линейки SSL/TLS, в то время как SSL 3.0 предоставил платформу для разработки последующих версий TLS. Протоколы SSL/TLS характеризуются порядком криптографических преобразований EtM.

Протокол SSL 3.0 включает рукопожатия создания и восстановления безопасных соединений, обеспечивает подписи RSA/DSA дайджестов MD5/SHA1 открытых криптографических параметров и асимметричное шифрование предварительных секретов. Расчёт главного секрета и ключевого материала обеспечен многократным вычислением хеш-функций MD5/SHA1 от случайных и константных величин. Аутентификация сообщений рукопожатия обеспечивается финальными сообщениями. Поддерживаются небезопасные криптографические алгоритмы DES, Fortezza, IDEA, MD5, RC2, RC4, SHA1, включая экспортные реализации. Режим работы блочных шифров представлен CBC.

Версия TLS 1.0 добавляет параметры расширения в протокол квитирования. Обычное хеширование заменено на функцию PRF, использующую HMAC на основе MD5/SHA1. Относительно SSL 3.0 протокол TLS 1.0 значительных архитектурных изменений не претерпел.

В TLS 1.1 введено новое форматирование сообщений в рамках защиты от атак Блейхенбахера. Для защиты от атак на режим CBC вводится явный вектор инициализации блочных шифров. В шифронаборы добавлен алгоритм AES-128/256, исключены экспортные реализации. В случае некорректного дополнения блочного шифра ошибка расшифрования заменена на оповещение сбоя MAC-кода.

Протокол TLS 1.2 делает акцент на задействование расширений, зачастую указывающих на поддерживаемые комбинации хеш-функций и связанных с ними алгоритмов подписи. Схема подписи RSA заменяет двойной дайджест MD5/SHA1 однократным. Сертификат открытого ключа одного алгоритма может подписываться с использованием другого алгоритма, например ключ RSA, подписанный схемой DSA. В отличие от TLS 1.0 и TLS 1.1, функция PRF TLS 1.2 явно согласуется криптонаборами, структурно упрощена использованием одного HMAC и отсутствием разделения секрета на части. По умолчанию PRF задается HMAC-SHA256. Исключены алгоритмы IDEA, DES. Рост безопасности TLS 1.2 обусловлен поддержкой эллиптических кривых, а также схем AEAD, представленных режимами CCM/GCM. Для финальных сообщений применяется хеш-функция, согласованная криптонаборами, ранние релизы SSL/TLS фиксировали MD5/SHA1. Отключено предупреждение об ошибке предварительного секрета, который рандомизируется в случае некорректности и обрабатывается неотличимым от правильно отформатированных сообщений способом, устраняя утечку информации.

Квитирование TLS 1.3 сокращено до трёх циклов, поддерживаемые расширения и данные после приветствия сервера защищены криптографически. Защита от атак на понижение версий обеспечивается включением фиксированных значений в случайные данные сервера, определённых отдельно для попыток согласования TLS 1.2 или TLS 1.1 и более ранних версий. Добавлена поддержка и согласование режимов обмена PSK, TLS 1.3 фиксирует KDF для связки с PSK. Функционал 0-RTT обеспечивает передачу защищенных ранних данных приложений на PSK до окончания квитирования. В рамках предотвращения атак с выбранным рандомным значением клиента на подписи ранних TLS изменена структура сообщений верификации сертификатов. Выработка ключевого материала TLS 1.3 использует более сложный механизм расчёта для большего количества секретов, где PRF заменена на HKDF. Перерасчёт ключей происходит после финишного сообщения. Финишные сообщения используют HMAC совместно с HKDF. Протокол TLS 1.3 чётко разграничивает механизмы аутентификации и обмена ключами от алгоритмов защиты записи, одна хэш-функция используется для вывода ключей и MAC. Данные приложений защищены исключительно режимами AEAD. Асимметричные алгоритмы определены (EC) DHE, RSA, ECDSA, EdDSA. Применение эллиптической криптографии ставит упор на использование фиксированной надёжной точки кривой взамен её согласования. Статические RSA и DH, алгоритм DSA и сертификаты OpenPGP, а также MD5 и SHA224 запрещены TLS 1.3. Дайджест SHA1 отмечен только для целей обратной совместимости.

Протокол RuTLS, построенный на основе TLS 1.3, корректирует логику взаимодействия участников соединения, используемые алгоритмы/криптонаборы и систему выработки ключей, призванные повысить безопасность системы. Наложены ограничения на использование 0-RTT, введены требования на порядок передачи данных приложений, генерации случайных чисел/криптографических ключей и используемые при этом идентификаторы. Изменения основаны на включении поддержки криптографических схем Лимонник-3, Крокус и Эхинацея-2(3) Российской Федерации.

Протокол IPsec прошёл три основных этапа развития. Протокол IKEv1 поддерживает DES, 3DES, IDEA, Blowfish, RC5, CAST в режиме CBC, а также MD5, SHA1 и Tiger в PRF. Методы аутентификации основаны на PSK, DSA, RSA, DH и ECC с использованием PRF, структура которой не определяется. Поддерживается аутентификация с одновременным использованием симметричных и асимметричных методов. Ключевой материал выводится функцией PRF на основе переданных идентификаторов, одноразовых номеров, выработанных секретов, значений DH и других параметров. Включение DH обеспечивает свойство PFS. Секретные ключи AH/ESP генерируются PRF по данным, полученным в предыдущих сообщениях.

Механизм аутентификации IKEv2 применяет общие ключи, PRF и опирается на единственное одноименное поле взамен распределения по всему обмену IKEv1. Зафиксированы требования к одноразовым номерам. Введены селекторы трафика, содержащие параметры пакетов *SA*. Добавлены расширяемые методы аутентификации EAP, однако без выработки общего ключа они небезопасны. В рамках борьбы с DoS-атаками описан пример реализации cookie-данных на основе одноразовых номеров, IP, *SPI* и секрета. Выработка ключевого материала фиксирует порядок вычисления секретов аутентификации и шифрования. Спецификация IKEv2 добавляет возможности работы через NAT. Нахождение за NAT идентифицируется на основе дайджеста SHA1 от идентификаторов участников взаимодействия. Введён способ определения динамического IP участника, расположенного за NAT. Реализована полноценная поддержка явного уведомления о перегрузке сети ECN. В криптонаборы добавлены алгоритмы AES в режимах CBC/CTR. Функции целостности и PRF представлены HMAC на основе MD5, SHA1, Tiger, AES-128-XCBC, DES-MAC, KPDK-MD5. Сжатие содержит DEFLATE, LZS и LZJH.

Протокол IKEv3 делает обязательным применение селекторов трафика, обновление ключевого материала всегда включает разделяемый секрет DH. Допускается произвольный порядок инкапсуляции полезной нагрузки. Идентификация *SA* использует всё сообщение и его дайджест. Разъяснено применение критического флага контроля типа полезной нагрузки. Введены требования предпочтительных длин ключей PRF. Даже при отсутствии NAT требуется поддержка обработки инкапсулированных и не инкапсулированных в UDP пакетов. Криптокомбинации разделены на комплекты с режимами AEAD и обычным шифрованием с отдельными алгоритмами целостности. Расширен список событий при аутентификации, различии состояния сторон, проверки MAC, в которых необходимы оповещения об ошибках. Преодоление коллизий и рассинхронизации состояний *SA* обеспечено добавлением уведомлений временной невозможности ответа на запрос и отсутствия дочерних ассоциаций безопасности.

Формирование заголовка AHv1 происходит за счёт хеш-функции, согласованной *SA*. Для расчёта *ICV* принимаются неизменяемые поля, модифицируемые поля обнуляются. Обязательна поддержка ключевого MD5, корректирующие коды исключены. Выпуск AHv2 стал самодостаточной платформой с интегрированной службой защиты от воспроизведения на основе порядковых номеров, обязательными функциями HMAC-MD5-96/HMAC-SHA1-96, фиксированными списками исключённых из расчёта *ICV* изменяемых данных IP. Выравнивание обеспечено паддингом, расположенным за *ICV*. Туннельный режим рассматривается как неотъемлемая часть AHv2. В AHv3 корректная обработка коллизий *SPI* между одноадресными и многоадресными архитектурами решена новым алгоритмом поиска *SA* в базе данных SAD. Новое расширение включает 64-разрядный порядковый номер *ESN* и способы его управления. Внесение паддинга больше достаточного минимума запрещено. Обязательны HMAC-

SHA1-96, AES-XCBC-MAC-96 и HMAC-MD5-96. Порядок криптографических преобразований IPsec соответствует E&M.

Обработка ESPv1 идейно схожа с AH. Целостность ESPv1 не описывается. Обязательным объявлен алгоритм DES-CBC. Версия ESPv2 описана как законченное решение. Добавлен 32-разрядный порядковый номер сообщений, управляемый аналогично AH. Вектор инициализации включается явно или неявно, заполнение используется для кратности размеру блока алгоритма шифрования. Поле *ICV* вычисляется от пакета ESPv2. Обязательными для поддержки являются DES-CBC с явным вектором инициализации, HMAC-MD5-96, HMAC-SHA1-96. Обеспечение целостности за счёт ESPv3 отмечено как более производительная альтернатива AH. Введена функция TFC для маскирования отсутствия реального трафика. Описана модель данных полезной нагрузки и *ICV* для работы с режимами AEAD, которые способны отдельно вычислять *ICV*. Механизм защиты от повторов использует ESN и совпадает с методами AHv3. Дополнение ограничено 255 байтами. Криптографические алгоритмы ESPv3 содержат 3DES-CBC, AES-CBC/CTR, а DES-CBC не рекомендуется. Целостность требует реализации HMAC-SHA1-96, AES-XCBC-MAC-96 и возможность HMAC-MD5-96. Заголовки ESP и AH можно комбинировать в разных вариантах.

В рассмотренных протоколах безопасности прослеживается тенденция отказа от применения нескольких слабо исследованных и нестойких криптографических функций, например для задач расчёта ключевого материала, обеспечения аутентификации и конфиденциальности, в пользу использования одного исследованного и надёжного алгоритма или криптографического параметра. Вместе с тем развитие криптопротоколов SSL/TLS и IPsec значительно увеличивает количество вариантов их состояний, характеризуемых комбинациями криптографических величин, применяемых алгоритмов, оповещений об ошибках и других параметров, что в значительной степени усложняет анализ безопасности системы.

ЛИТЕРАТУРА

1. *Kipp E. B. H.* The SSL Protocol. Netscape Communications Corp., 1995 (Expires 10 / 95). 26 p. `https://tools.ietf.org/html/draft-hickman-netscape-ssl-00`.
2. *Polk T. and Turner S.* Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176. Internet Engineering Task Force (IETF), 2011. 4 p. `https://tools.ietf.org/html/rfc6176`.
3. *Freier A., Karlton P., and Kocher P.* The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101. Internet Engineering Task Force (IETF), 2011. 67 p. `https://tools.ietf.org/html/rfc6101`.
4. *Allen C. and Dierks T.* The TLS Protocol Version 1.0. RFC 2246. Network Working Group, 1999. 80 p. `https://tools.ietf.org/html/rfc2246`.
5. *Kaliski B.* PKCS#1: RSA Encryption Standard, version 1.5. RFC 2313. Network Working Group, 1998. 19 p. `https://tools.ietf.org/html/rfc2313`.
6. *Dierks T. and Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346. Network Working Group, 2006. 87 p. `https://tools.ietf.org/html/rfc4346`.
7. *Jonsson J. and Kaliski B.* Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447. Network Working Group, 2003. 72 p. `https://tools.ietf.org/html/rfc3447`.
8. *Ford W., Housley R., Polk W., and Solo D.* Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280. Network Working Group, 2002. 129 p. `https://tools.ietf.org/html/rfc3280`.

9. http://www.openssl.org/~bodo/tls-cbc.txt — Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures, 2004.
10. *Dierks T. and Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. Network Working Group, 2008. 104 p. https://tools.ietf.org/html/rfc5246.
11. *Eastlake D. 3rd.* Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066. Internet Engineering Task Force (IETF), 2011. 25 p. https://tools.ietf.org/html/rfc6066.
12. *Dworkin M.* Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST Special Publication 800-38C, 2004. 27 p.
13. *Dworkin M.* Recommendation for Block Cipher Modes of Operation: Galois / Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007. 39 p.
14. *McGrew D.* An Interface and Algorithms for Authenticated Encryption. RFC 5116. Network Working Group, 2008. 22 p. https://tools.ietf.org/html/rfc5116.
15. *Mavrogiannopoulos N.* Using OpenPGP Keys for Transport Layer Security (TLS) Authentication. RFC 5081. Network Working Group, 2007. 8 p. https://tools.ietf.org/html/rfc5081.
16. *Blake-Wilson S., Bolyard N., Gupta V., et al.* Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492. Network Working Group, 2006. 35 p. https://tools.ietf.org/html/rfc4492.
17. *Bleichenbacher D.* Chosen ciphertext attacks against protocols based on RSA Encryption Standard PKCS#1 // CRYPTO'98. LNCS. 1998. V. 1462. P. 1–12.
18. *Klima V., Pokorny O., and Rosa T.* Attacking RSA-based Sessions in SSL/TLS. Cryptology ePrint Archive: Report 2003/052, 2003. 23 p.
19. https://csrc.nist.gov/publications/detail/fips/186/3/archive/2009-06-25 — Digital Signature Standard (DSS). NIST FIPS PUB 186-3, 2009. 131 p.
20. *Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. Internet Engineering Task Force (IETF), 2018. 160 p. https://tools.ietf.org/html/rfc8446.
21. *Eronen P. and Krawczyk H.* HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869. Internet Engineering Task Force (IETF), 2010. 14 p. https://www.rfc-editor.org/info/rfc5869.
22. https://standards.ieee.org/standard/1363-2000.html — IEEE Standard Specifications for Public Key Cryptography. IEEE Std 1363-2000, 2000. 236 p.
23. *Hamburg M., Langley A., and Turner S.* Elliptic Curves for Security. RFC 7748. Internet Research Task Force (IRTF), 2016. 22 p. https://tools.ietf.org/html/rfc7748.
24. *Гребнев С. В., Лазарева Е. В., Лебедев П. А. и др.* Интеграция отечественных протоколов выработки общего ключа в протокол TLS 1.3 // Прикладная дискретная математика. Приложение. 2018. № 11. С. 62–65.
25. *Матюхин Д. В.* О некоторых свойствах схем выработки общего ключа, использующих инфраструктуру открытых ключей, в контексте разработки стандартизированных криптографических решений. 2011. https://www.ruscrypto.ru/resource/archive/rc2011/files/02_matyukhin.pdf.
26. *Нестеренко А. Ю.* Об одном подходе к построению защищенных соединений // Математические вопросы криптографии. 2013. № 4:2. С. 101–111.
27. *Гребнев С. В.* О возможности стандартизации протоколов выработки общего ключа. РусКрипто, М., 2014. https://www.ruscrypto.ru/resource/archive/rc2014/files/03_grebnev.pdf.
28. *Carrel D. and Harkins D.* The Internet Key Exchange (IKE). RFC 2409. Network Working Group, 1998. 41 p. https://tools.ietf.org/html/rfc2409.

29. *Orman H.* The Oakley Key Determination Protocol. RFC 2412. Network Working Group, 1998. 55 p. https://tools.ietf.org/html/rfc2412.
30. *Krawczyk H.* SKEME: A versatile secure key exchange mechanism for Internet // Proc. Internet Society Symp. on Network and Distributed Systems Security, San Diego, CA, USA, 1996. P. 114–127.
31. *Maughhan D., Schertler M., Schneider M., and Turner J.* Internet Security Association and Key Management Protocol (ISAKMP). RFC 2408. 1998. https://tools.ietf.org/html/rfc2408.
32. *Schneier B.* Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd ed. N.Y.: Wiley, 1996. 783 p.
33. *Kaufman C.* Internet Key Exchange (IKEv2) Protocol. RFC 4306. Network Working Group, 2005. 99 p. https://tools.ietf.org/html/rfc4306.
34. *Piper D.* The Internet IP Security Domain of Interpretation for ISAKMP. RFC 2407. Network Working Group, 1998. 32 p. https://tools.ietf.org/html/rfc2407.
35. *Aboba B., Blunk L., Carlson J., et al.* Extensible Authentication Protocol (EAP). RFC 3748. Network Working Group, 2004. 67 p. https://tools.ietf.org/html/rfc3748.
36. *Asokan N., Nierni V., and Nyberg K.* Man-in-the-Middle in Tunneled Authentication Protocols. Cryptology ePrint Archive: Report 2002/163, 2002. 15 p.
37. *Monsour B., Pereira R., Shacham A., and Thomas M.* IP Payload Compression Protocol (IPComp). RFC 3173. Network Working Group, 2001. 13 p. https://tools.ietf.org/html/rfc3173.
38. *DiBurro L., Huttunen A., Stenberg M., et al.* UDP Encapsulation of IP Security ESP Packets. RFC 3948. Network Working Group, 2005. 15 p. https://tools.ietf.org/html/rfc3948.
39. *Black D., Floyd S., and Ramakrishnan K.* The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168. Network Working Group, 2001. 63 p. https://tools.ietf.org/html/rfc3168.
40. *Kent S. and Seo K.* Security Architecture for the Internet Protocol. RFC 4301. Network Working Group, 2005. 101 p. https://tools.ietf.org/html/rfc4301.
41. *Eronen P., Hoffman P., Kaufman C., and Nir Y.* Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996. Internet Engineering Task Force (IETF), 2010. 138 p. https://tools.ietf.org/html/rfc5996.
42. *Eronen P. and Hoffman P.* IKEv2 Clarifications and Implementation Guidelines. RFC 4718. Network Working Group, 2006. 58 p. https://tools.ietf.org/html/rfc4718.
43. *Berners-Lee T., Fielding R., Frystyk H., et al.* Hypertext Transfer Protocol — HTTP / 1.1. RFC 2616. Network Working Group, 1999. 176 p. https://tools.ietf.org/html/rfc2616.
44. *Eronen P., Laganier J., and Madson C.* IPv6 Configuration in Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5739. Internet Engineering Task Force (IETF), 2010. 32 p. https://tools.ietf.org/html/rfc5739.
45. *Atkinson R.* The IP Authentication Header. RFC 1826. Network Working Group, 1995. 13 p. https://tools.ietf.org/html/rfc1826.
46. *Metzger P. and Simpson W.* IP Authentication with Keyed MD5. RFC 1828. Network Working Group, 1995. 5 p. https://tools.ietf.org/html/rfc1828.
47. *Atkinson R. and Kent S.* IP Authentication Header. RFC 2402. Network Working Group, 1998. 22 p. https://tools.ietf.org/html/rfc2402.
48. *Kent S.* IP Authentication Header. RFC 4302. Network Working Group, 2005. 34 p. https://tools.ietf.org/html/rfc4302.

49. *Eastlake D. 3rd.* Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). RFC 4305. Network Working Group, 2005. 9 p. https://tools.ietf.org/html/rfc4305.
50. *Atkinson R.* IP Encapsulating Security Payload (ESP). RFC 1827. Network Working Group, 1995. 12 p. https://tools.ietf.org/html/rfc1827.
51. *Atkinson R. and Kent S.* IP Encapsulating Security Payload (ESP). RFC 2406. Network Working Group, 1998. 22 p. https://tools.ietf.org/html/rfc2406.
52. *Kent S.* IP Encapsulating Security Payload (ESP). RFC 4303. Network Working Group, 2005. 44 p. https://tools.ietf.org/html/rfc4303.

REFERENCES

1. *Kipp E. B. H.* The SSL Protocol. Netscape Communications Corp., 1995 (Expires 10 / 95). 26 p. https://tools.ietf.org/html/draft-hickman-netscape-ssl-00.
2. *Polk T. and Turner S.* Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176. Internet Engineering Task Force (IETF), 2011. 4 p. https://tools.ietf.org/html/rfc6176.
3. *Freier A., Karlton P., and Kocher P.* The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101. Internet Engineering Task Force (IETF), 2011. 67 p. https://tools.ietf.org/html/rfc6101.
4. *Allen C. and Dierks T.* The TLS Protocol Version 1.0. RFC 2246. Network Working Group, 1999. 80 p. https://tools.ietf.org/html/rfc2246.
5. *Kaliski B.* PKCS#1: RSA Encryption Standard, version 1.5. RFC 2313. Network Working Group, 1998. 19 p. https://tools.ietf.org/html/rfc2313.
6. *Dierks T. and Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346. Network Working Group, 2006. 87 p. https://tools.ietf.org/html/rfc4346.
7. *Jonsson J. and Kaliski B.* Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447. Network Working Group, 2003. 72 p. https://tools.ietf.org/html/rfc3447.
8. *Ford W., Housley R., Polk W., and Solo D.* Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280. Network Working Group, 2002. 129 p. https://tools.ietf.org/html/rfc3280.
9. http://www.openssl.org/~bodo/tls-cbc.txt — Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures, 2004.
10. *Dierks T. and Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. Network Working Group, 2008. 104 p. https://tools.ietf.org/html/rfc5246.
11. *Eastlake D. 3rd.* Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066. Internet Engineering Task Force (IETF), 2011. 25 p. https://tools.ietf.org/html/rfc6066.
12. *Dworkin M.* Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST Special Publication 800-38C, 2004. 27 p.
13. *Dworkin M.* Recommendation for Block Cipher Modes of Operation: Galois / Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007. 39 p.
14. *McGrew D.* An Interface and Algorithms for Authenticated Encryption. RFC 5116. Network Working Group, 2008. 22 p. https://tools.ietf.org/html/rfc5116.
15. *Mavrogiannopoulos N.* Using OpenPGP Keys for Transport Layer Security (TLS) Authentication. RFC 5081. Network Working Group, 2007. 8 p. https://tools.ietf.org/html/rfc5081.

16. *Blake-Wilson S., Bolyard N., Gupta V., et al.* Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492. Network Working Group, 2006. 35 p. https://tools.ietf.org/html/rfc4492.
17. *Bleichenbacher D.* Chosen ciphertext attacks against protocols based on RSA Encryption Standard PKCS#1. CRYPTO'98, LNCS, 1998, vol. 1462, pp. 1–12.
18. *Klima V., Pokorny O., and Rosa T.* Attacking RSA-based Sessions in SSL/TLS. Cryptology ePrint Archive: Report 2003/052, 2003. 23 p.
19. https://csrc.nist.gov/publications/detail/fips/186/3/archive/2009-06-25 — Digital Signature Standard (DSS). NIST FIPS PUB 186-3, 2009. 131 p.
20. *Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. Internet Engineering Task Force (IETF), 2018. 160 p. https://tools.ietf.org/html/rfc8446.
21. *Eronen P. and Krawczyk H.* HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869. Internet Engineering Task Force (IETF), 2010. 14 p. https://www.rfc-editor.org/info/rfc5869.
22. https://standards.ieee.org/standard/1363-2000.html — IEEE Standard Specifications for Public Key Cryptography. IEEE Std 1363-2000, 2000. 236 p.
23. *Hamburg M., Langley A., and Turner S.* Elliptic Curves for Security. RFC 7748. Internet Research Task Force (IRTF), 2016. 22 p. https://tools.ietf.org/html/rfc7748.
24. *Grebnev S. V., Lazareva E. V., Lebedev P. A., et al.* Integratsiya otechestvennykh protokolov vyrabotki obshchego klyucha v protokol TLS 1.3 [Integration of domestic shared key generation protocols into TLS 1.3]. Prikladnaya Diskretnaya Matematika. Prilozhenie, 2018, no. 11, pp. 62–65. (in Russian)
25. *Matyukhin D. V.* O nekotorykh svoystvakh skhem vyrabotki obshchego klyucha, ispol'zuyushchikh infrastrukturu otkrytykh klyuchey, v kontekste razrabotki standartizirovannykh kriptograficheskikh resheniy [On some properties of public key generation schemes that use the public key infrastructure in the context of developing standardized cryptographic solutions]. 2011. https://www.ruscrypto.ru/resource/archive/rc2011/files/02_matyukhin.pdf. (in Russian)
26. *Nesterenko A. Y.* Ob odnom podkhode k postroyeniyu zashchishchennykh soyedineniy [About one approach to building secure connections]. Matematicheskiye Voprosy Kriptografii, 2013, no. 4:2, pp. 101–111. (in Russian)
27. *Grebnev S. V.* O vozmozhnosti standartizatsii protokolov vyrabotki obshchego klyucha [On the possibility of standardizing protocols for generating a shared key]. RusKripto, Moscow, 2014. https://www.ruscrypto.ru/resource/archive/rc2014/files/03_grebnev.pdf. (in Russian)
28. *Carrel D. and Harkins D.* The Internet Key Exchange (IKE). RFC 2409. Network Working Group, 1998. 41 p. https://tools.ietf.org/html/rfc2409.
29. *Orman H.* The Oakley Key Determination Protocol. RFC 2412. Network Working Group, 1998. 55 p. https://tools.ietf.org/html/rfc2412.
30. *Krawczyk H.* SKEME: A versatile secure key exchange mechanism for Internet. Proc. Internet Society Symp. on Network and Distributed Systems Security, San Diego, CA, USA, 1996, pp. 114–127.
31. *Maughhan D., Schertler M., Schneider M., and Turner J.* Internet Security Association and Key Management Protocol (ISAKMP). RFC 2408. 1998. https://tools.ietf.org/html/rfc2408.
32. *Schneier B.* Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd ed. N.Y., Wiley, 1996. 783 p.

33. *Kaufman C.* Internet Key Exchange (IKEv2) Protocol. RFC 4306. Network Working Group, 2005. 99 p. https://tools.ietf.org/html/rfc4306.
34. *Piper D.* The Internet IP Security Domain of Interpretation for ISAKMP. RFC 2407. Network Working Group, 1998. 32 p. https://tools.ietf.org/html/rfc2407.
35. *Aboba B., Blunk L., Carlson J., et al.* Extensible Authentication Protocol (EAP). RFC 3748. Network Working Group, 2004. 67 p. https://tools.ietf.org/html/rfc3748.
36. *Asokan N., Nierni V., and Nyberg K.* Man-in-the-Middle in Tunneled Authentication Protocols. Cryptology ePrint Archive: Report 2002/163, 2002. 15 p.
37. *Monsour B., Pereira R., Shacham A., and Thomas M.* IP Payload Compression Protocol (IPComp). RFC 3173. Network Working Group, 2001. 13 p. https://tools.ietf.org/html/rfc3173.
38. *DiBurro L., Huttunen A., Stenberg M., et al.* UDP Encapsulation of IP Security ESP Packets. RFC 3948. Network Working Group, 2005. 15 p. https://tools.ietf.org/html/rfc3948.
39. *Black D., Floyd S., and Ramakrishnan K.* The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168. Network Working Group, 2001. 63 p. https://tools.ietf.org/html/rfc3168.
40. *Kent S. and Seo K.* Security Architecture for the Internet Protocol. RFC 4301. Network Working Group, 2005. 101 p. https://tools.ietf.org/html/rfc4301.
41. *Eronen P., Hoffman P., Kaufman C., and Nir Y.* Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996. Internet Engineering Task Force (IETF), 2010. 138 p. https://tools.ietf.org/html/rfc5996.
42. *Eronen P. and Hoffman P.* IKEv2 Clarifications and Implementation Guidelines. RFC 4718. Network Working Group, 2006. 58 p. https://tools.ietf.org/html/rfc4718.
43. *Berners-Lee T., Fielding R., Frystyk H., et al.* Hypertext Transfer Protocol — HTTP / 1.1. RFC 2616. Network Working Group, 1999. 176 p. https://tools.ietf.org/html/rfc2616.
44. *Eronen P., Laganier J., and Madson C.* IPv6 Configuration in Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5739. Internet Engineering Task Force (IETF), 2010. 32 p. https://tools.ietf.org/html/rfc5739.
45. *Atkinson R.* The IP Authentication Header. RFC 1826. Network Working Group, 1995. 13 p. https://tools.ietf.org/html/rfc1826.
46. *Metzger P. and Simpson W.* IP Authentication with Keyed MD5. RFC 1828. Network Working Group, 1995. 5 p. https://tools.ietf.org/html/rfc1828.
47. *Atkinson R. and Kent S.* IP Authentication Header. RFC 2402. Network Working Group, 1998. 22 p. https://tools.ietf.org/html/rfc2402.
48. *Kent S.* IP Authentication Header. RFC 4302. Network Working Group, 2005. 34 p. https://tools.ietf.org/html/rfc4302.
49. *Eastlake D. 3rd.* Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). RFC 4305. Network Working Group, 2005. 9 p. https://tools.ietf.org/html/rfc4305.
50. *Atkinson R.* IP Encapsulating Security Payload (ESP). RFC 1827. Network Working Group, 1995. 12 p. https://tools.ietf.org/html/rfc1827.
51. *Atkinson R. and Kent S.* IP Encapsulating Security Payload (ESP). RFC 2406. Network Working Group, 1998. 22 p. https://tools.ietf.org/html/rfc2406.
52. *Kent S.* IP Encapsulating Security Payload (ESP). RFC 4303. Network Working Group, 2005. 44 p. https://tools.ietf.org/html/rfc4303.