Построение защищенных каналов связи

Макаров Артём

МИФИ 2020

Защищенный канал связи

- Обеспечить аутентичность и секретность сообщений при передаче по каналу связи при активном противнике
- Возможности противника – любые эффективные взаимодействия с каналом, включая повтор и модификацию передаваемых сообщений, возможность разовой компрометации сессионных ключей, возможность имперсонификации источника сообщений
- Цели противника – нарушение ИБ передаваемых сообщений

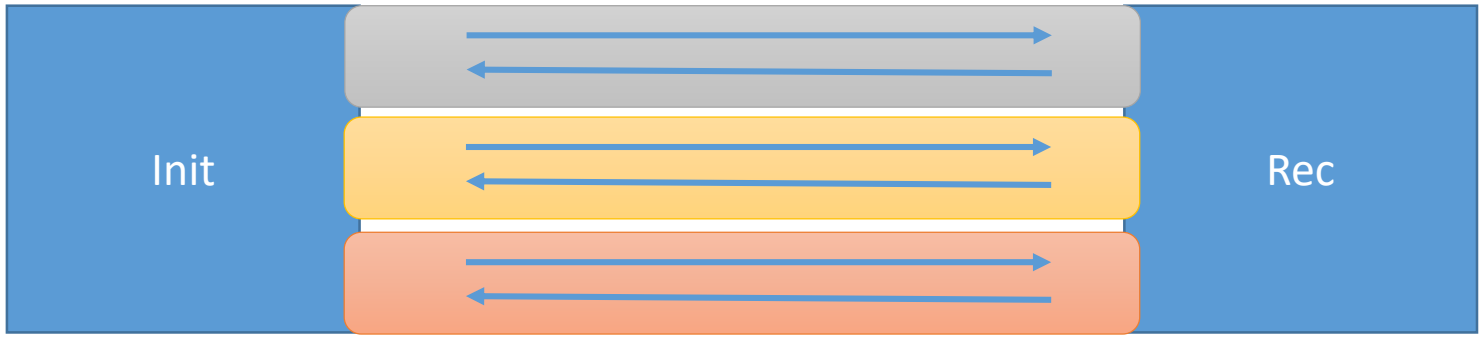
Защищенный канал связи

2 основных стороны – инициатор (Init) и получатель (Rec)

Нулевая фаза – согласование параметров

Первая фаза – согласование сессионного ключа

Вторая фаза – обмен сообщениями через защищенный канал

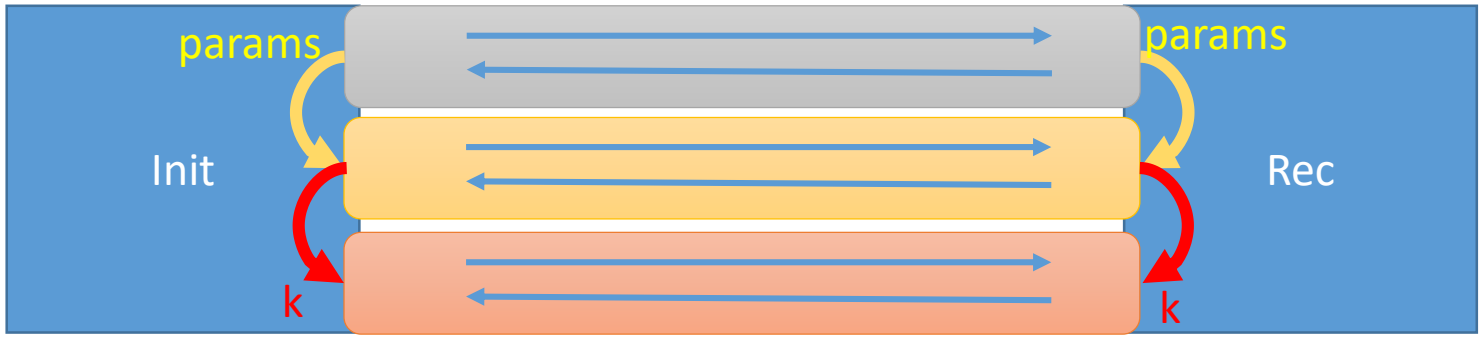


Защищенный канал связи

Нулевая фаза может отсутствовать, если параметры заранее известны сторонам. Результат – согласованные параметры на обоих сторонах.

Цель первой фазы выработка общего симметричного ключа, как правило асимметричным способом из общего ключевого материала (согласование ключа, Key establishment).

Третья фаза как правило осуществляется с помощью симметричной криптографии (AEAD).



Аутентификация в каналах связи

Очевидно, что для построения защищенного канала связи необходимо построение отношения доверия между сторонами (иначе стороны не будут знать, с кем они общаются).

Отношение доверия бывает односторонним и двухсторонним.

Одностороннее отношение доверия аутентифицирует только получателя.

Двухстороннее отношение доверия аутентифицирует и получателя и отправителя.

Аутентификация в каналах связи

Криптографическая аутентификация должна производится на основе знания некоторого секрета, неизвестного противнику.

- Аутентифицируемая сторона должна иметь некоторый долгоживущий секрет
- Аутентифицирующая сторона должна иметь возможность проверить владение секретом
- Аутентифицирующая сторона должна иметь возможность сопоставить наличие секрета другой стороной с её идентичностью

Аутентификация с использованием асимметричной криптографии

- Секрет – закрытый статический долговременный ключ
- Проверка секрета – доказательство владения секретным ключом, путём его использования
- Привязка идентичности к открытому ключу

Возникает вопрос обмена открытыми ключами.

Решения:

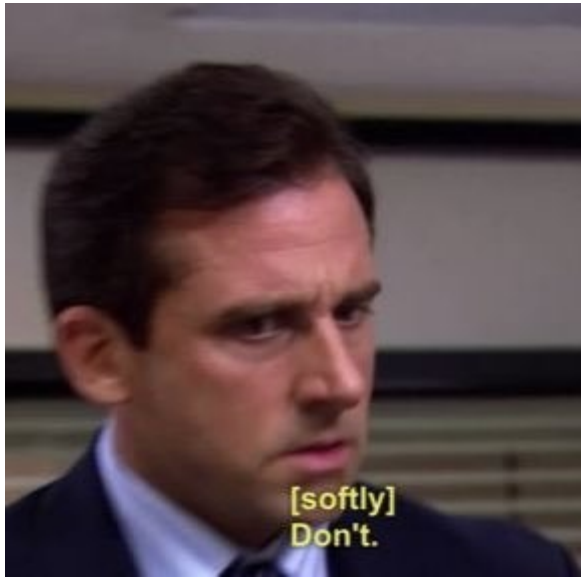
- Использование удостоверяющего центра
- Прямой обмен ключами до протокольного взаимодействия

Некоторые требования к протоколам согласования ключей

- Аутентичность сущности (entity authentication) – кто то действует от лица конкретной сущности в ходе протокола
- Аутентичность источника (data origin authentication) – сообщения приходят от конкретного источника
- (неявная) Аутентичность ключа (implicit key authentication) – никто, кроме определённой сущности не может получить секретный ключ
- Подтверждение ключа (key confirmation) – возможность подтверждения какой-либо другой стороной факта владения выработанным секретным ключом
- Явная аутентичность ключа (explicit key authentication) – возможность подтверждения конкретной другой стороной факта обладания выработанным секретным ключом (неявная аутентичность + подтверждение ключа)

Некоторые модели стойкости протоколов

- Защита от чтения назад (forward secrecy, т.е. «дальнейшая стойкость») – компрометация **долговременных** ключей не приводит к компрометации прошлых **сессионных ключей**
- Защита от чтения вперёд (future secrecy, т.е. «будущая стойкость») – компрометация **сессионных ключей** не приводит к компрометации **будущих сессионных ключей**



- Rule #1 of protocol design: **Don’t**
  - not even by simplifying existing protocols

The Cryptographic Doom Principle

- «When it comes to designing secure protocols, I have a principle that goes like this: if you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will somehow inevitably lead to doom.»

- https://moxie.org/2011/12/13/the-cryptographic-doom-principle.html

Padding oracle

- TLS до 1.2 включительно использовал mac-then-encrypt (rand. AES CBC + MAC), что в ряде сценариев приводило к атакам типа Padding oracle
- Основная проблема – расшифрование в mac-then-encrypt должно осуществляться до проверки аутентичности.

SSL 3.0

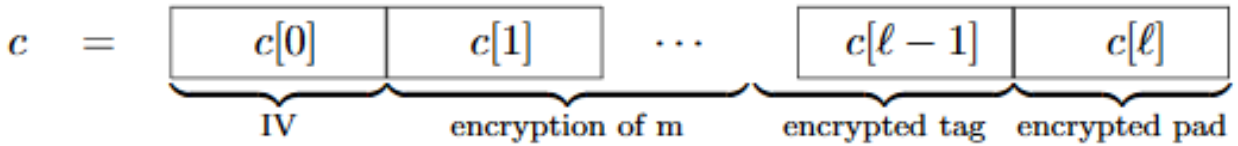
Пусть используется AES в CBC режиме.

Шифрование:

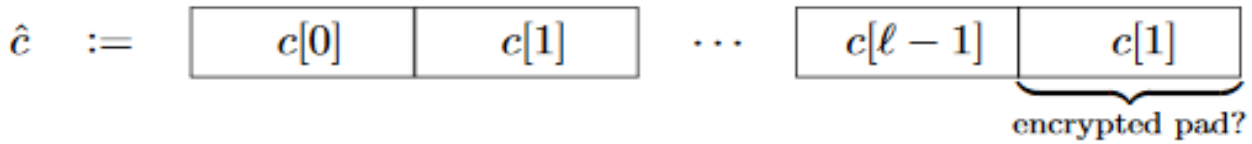
- Вычисляется MAC для сообщения.
- Дополнение. Если требуется $p > 0$ байтов для дополнения для сообщения и MAC, используется $p - 1$ случайный байт, а последний байт устанавливается в значение $(p - 1)$. Если сообщение уже необходимой длины – добавляется новый блок.
- Шифрование вычисляется на дополненном открытом тексте и MAC

Атака на SSL 3.0 (предполагая случайный IV)

Пусть противник получил некоторый шифртекст $c = E((k_e, k_m), m)$ для некоторого неизвестного сообщения m. Пусть длина сообщения такова, что сообщение и MAC дополняются полным блоком дополнения. Тогда шифртекст выглядит следующим образом:



Противник создаёт новый шифртекст c', заменяя последний блок на $c[1]$



Атака на SSL 3.0 (предполагая случайный IV)

При расшифровании последнего блока получатель имеет:

$$v = D(k_c, c[1]) \oplus c[l-1] = m[0] \oplus c[0] \oplus c[l-1]$$

Если последний байт равен 15, то весь последний блок будет отброшен как дополнение. Оставшаяся часть отрытого текста образует корректную пару открытый текст – MAC и сервер не сообщит об ошибке.

Если последний байт не равен 15, то часть последнего блока будет интерпретироваться как MAC, в результате сервер вернёт $\perp$.

Атака на SSL 3.0 (предполагая случайный IV)

Итого, если сервер не вернул $\perp$, тогда противник узнаёт, что последний байт $m[0]$ равен последнему байту $u = 15 \oplus c[0] \oplus c[l-1]$. Таким образом противник вычисляет байт открытого текста и нарушает семантическую стойкость.

Атаки данного типа носят называние padding oracle attack – т.е. имея оракул дополнения, который сообщает противнику корректно ли дополнение, противник осуществляет атаку.

TLS 1.3

RFC 8446

Цели

- Удаление legacy
- Увеличение производительности
- Уменьшение набора возможных атак

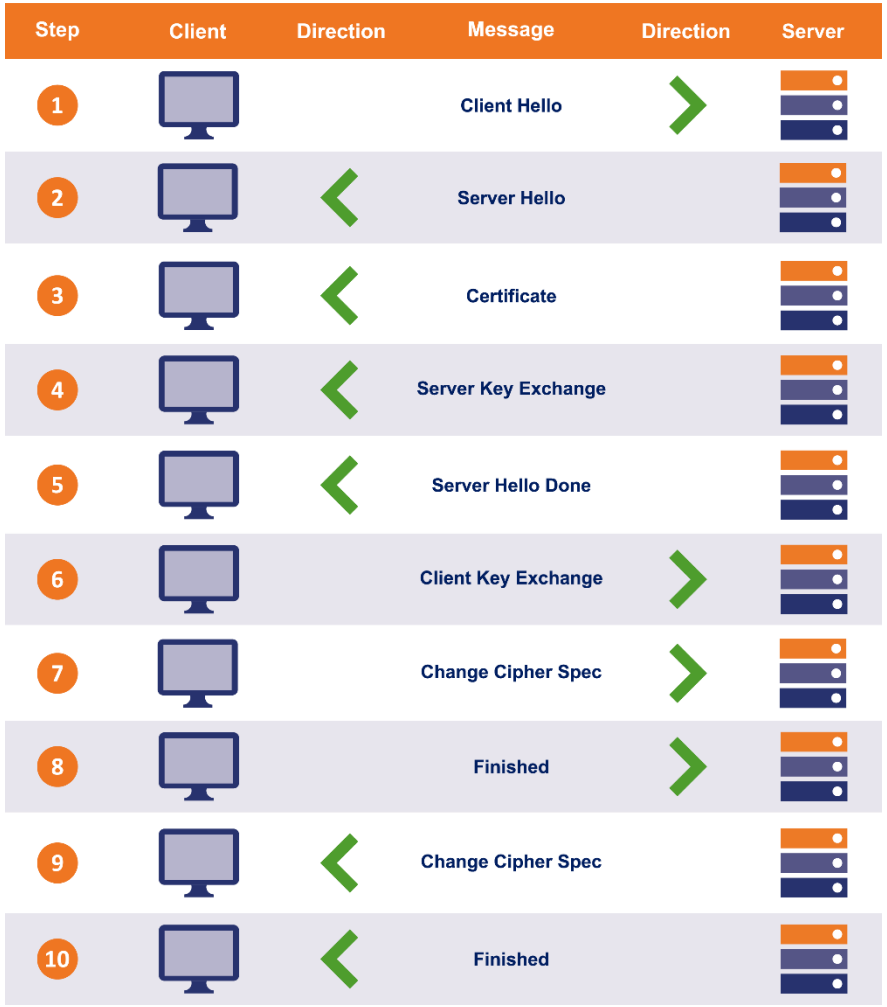
Согласование ключей в TSL 1.3

- Только эфемерный Диффи-Хеллман (удалили RSA)
- Только стойкий в настоящий момент набор параметров (определяющих группу).

Аутентифицированное шифрование

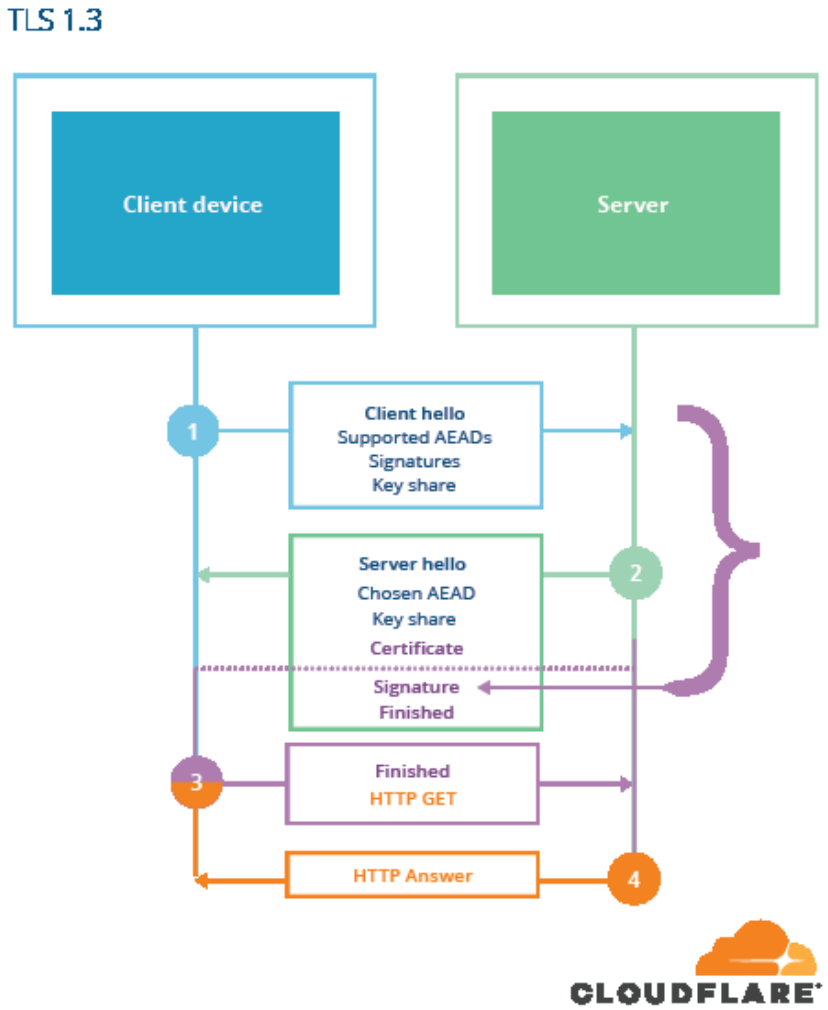
- Удаление проблемных и нестойких шифров (CBC, RC4)
- Разрешены только AEAD шифры

TLS 1.2 vs TLS 1.3 handshake



Электронные подписи

- Дополнение RSA-PSS (убрали PKCS#1 1.5)
- Сервер подписывает handshake (защищаясь от downgrade attack, когда in-the-middle противник навязывает сторонам нестойкие наборы)



Наборы алгоритмов

До версии 1.3 использовались наборы (cipher-suit) алгоритмов, определяющие используемые примитивы. Пример - ECDHE-ECDSA-AES-GCM-SHA256.

Версия 1.3 выделяет 3 независимых части названия

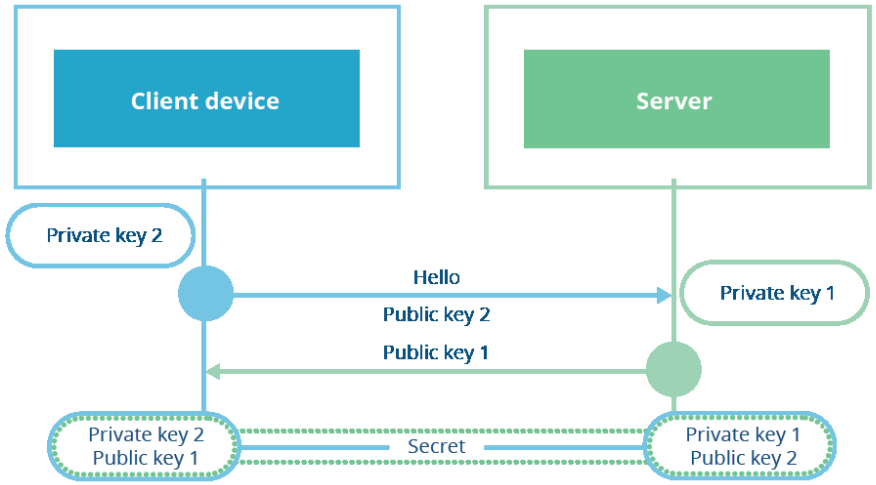
- Шифр + HKDF
- Протокол обмена ключами
- Алгоритм подписи

Пример – TLS_AES_128_GCM_SHA256, ECDHE, RSA_2048_SHA256

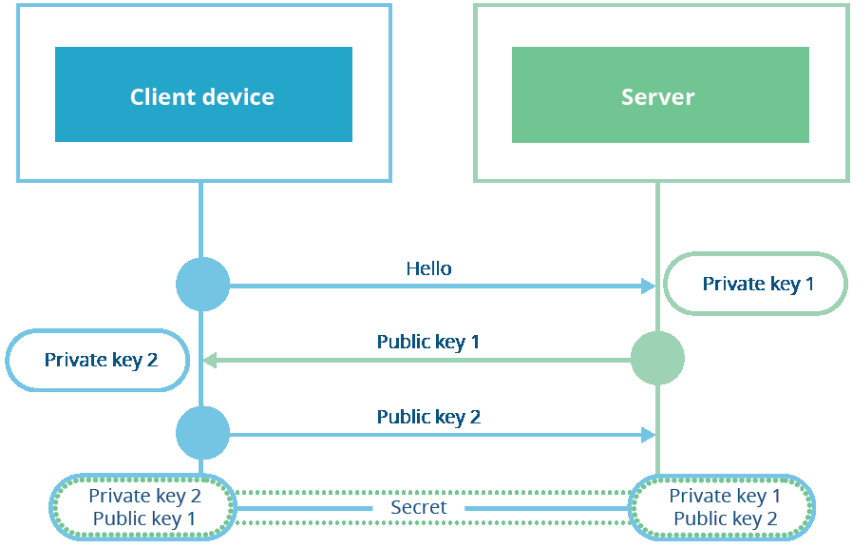
TLS 1.3 vs 1.2 public key Handshake

В 1.3 клиент может сразу отправить свой открытый ключ, не дожидаясь ответа от сервера со списком поддерживаемых алгоритмов (1-RTT).

DH 1.3 handshake



DH 1.2 handshake



TSL 1.3 ссылки

- https://tls13.ulfheim.net/
- https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/
- https://davidwong.fr/tls13/

- Отображение suit’ов браузера https://clienttest.ssllabs.com:8443/ssltest/viewMyClient.html
- Краткий разбор пакетов 1.2 https://medium.com/@ethicalevil/tls-handshake-protocol-overview-a39e8eee2cf5

Noise

Каркасная модель построения протоколов (Protocol framework)

Основная идея – построения tls-образных протоколов с необходимым набором свойств.

Используемые примитивы – ECDH 25519/448 (Согласование ключей), SHA256/512 / BLAKE2b/BLAKE2s (hash, построение PRF в HKDF), AES-GCM/ChaChaPoly (AEAD).

Протоколы Noise

2 основных этапа – согласование ключа и использование ключа в защищенном канале связи

Согласование ключа – Диффи-Хеллман + HKDF

Защищенный канал – AEAD шифрование

Протоколы Noise

- Каждый протокол состоит из шаблонов
- шаблон описываем передаваемые сообщения в виде одного или нескольких токенов
- Каждая сторона протокола обрабатывает последовательность токенов. Сначала постностью происходит обработка на передающей стороне.
- После каждого обработки каждой строки в паттерне отправителем может быть передано сообщение, для защиты которого используется текущий симметричный ключ, при его наличии

Токены Noise

- "e" – сгенерировать и передать эфемерный ключ Диффи-Хеллмана / получить эфемерный ключ Диффи-Хеллмана
- "s" – передать статический ключ Диффи-Хеллмана / получить эфемерный ключ Диффи-Хеллмана
- "ee" – использовать свой эфемерный ключ и эфемерный ключ получателя для выработки симметричного ключа. Данный ключ используется для AEAD шифрование всех передаваемый в дальнейшем данных, пока не будет выработан новый ключ
- "es",“se","ss", - аналогично “ee”, но для комбинаций эфемерный-статический ключ. Первая буква в токене означает ключ передающей стороны.
- "psk“ – использование пред-согласованного симметрично ключа для выработки сессионного ключа. Полученный ключ используется для AEAD шифрование всех передаваемый в дальнейшем данных, пока не будет выработан новый ключ

Пример Классический Диффи-Хеллман

- -> e
- <- e, ee

Строка 1.

Шаг 1. Генерация и отправка эфемерного ключа Ae->B

Шаг 2. Сторона B получает эфемерный ключ Ae

Строка 2.

Шаг 3. Генерация и отправка эфемерного ключа Be->A

Шаг 4. Выработка сессионного ключа из Ae и Be на стороне B

Шаг 5. Сторона A получает эфемерный ключ Bb

Шаг 6. Выработка сессионного ключа из Ae и Be на стороне B

Пример Диффи-Хеллман с односторонней и двухсторонней аутентификацией

- -> e
- <- e, ee, s, es

- -> e
- <- e, ee, s, es
- -> s, se

Именование паттернов

Односторонние шаблоны реализуют неинтерактивное согласование ключа, при котором инициатор передаёт единственное сообщение

A():

<pre-messages>

...

<messages>

где A - имя шаблона, pre-message - сообщения, передаваемые в шаблоне пред-сообщения, message - сообщения передаваемые в шаблоне сообщения.

Именование Шаблонов

- N: отсутствие статического ключа инициатора
- K: статический ключ инициатора известен получателю
- X: статический ключ инициатора передаётся получателю

Токены "s", "e", "e, s" внутри скобок означают что инициатор инициирован со следующими статическими / эфемерными ключами. Токены "rs", "re","re”, “rs" обозначают инициатор инициирован со знанием следующих открытых статических / эфемерных ключей получателя.

Пример:

N(rs):

<- s

...

-> e, es

Именование шаблонов

Интерактивные шаблоны состоят из одного или нескольких шаблонов сообщений, передаваемых между получателем и инициатором.

AA():

<pre-messages>

...

<messages>

где AA - имя шаблона, pre-message - сообщения, передаваемые в шаблоне пред-сообщения, message - сообщения передаваемые в шаблоне.

Именование шаблонов

Первая буква в имени шаблона определяет статический ключ инициатора, и может принимать одно из следующих значений:

- N: отсутствие статического ключа инициатора
- K: статический ключ инициатора известен получателю
- X: Статический ключ инициатора передаётся получателю
- I: Статический ключ инициатора немедленно передаётся получателю, даже в условиях отсутствия зашиты идентичности.

Вторая буква в имени шаблона определяет статический ключ получателя, и может принимать одно из следующих значений:

- N: отсутствие статического ключа получателя
- K: статический ключ получателя известен инициатору
- X: статический ключ получателя передаётся инициатору.

Именование паттернов

Пример

IK(s, rs):

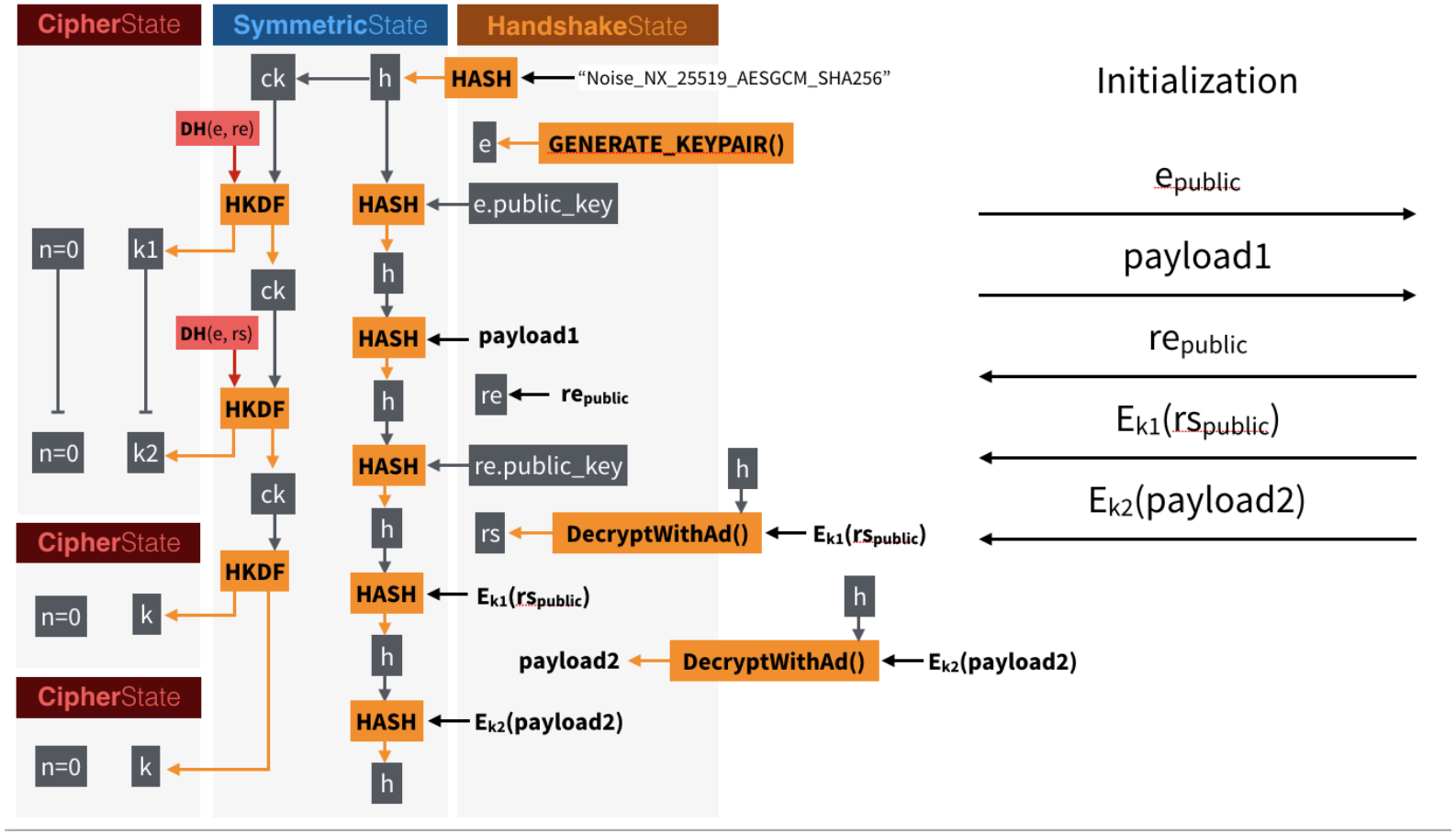
<- s

...

-> e, es, s, ss

<- e, ee, se

Функционирование протокола Noise



Больше про noise

- http://cryptowiki.net/index.php?title=%D0%9A%D0%B0%D1%80%D0%BA%D0%B0%D1%81%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%BF%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D0%BE%D0%B2_Noise
- http://cryptowiki.net/index.php?title=Noise_Protocol_Framework
- https://noiseprotocol.org/
- https://www.nccgroup.com/us/about-us/newsroom-and-events/blog/2018/august/introducing-disco/

WireGuard

Черновик стандарта построения VPN соединений.

- Все пакеты аутентифицированы, неаутентифицированные отбрасываются. Не отвечать на неаутентифицированные пакеты.
- 1 RRT handshake (Noise IK)
- Использование метки времени TAI64N в первом сообщении
- Опциональное использование psk для смягчения угрозы перед квантовым противником
- Использование cookie для смягчения DOS

Маршрутизация WireGuard

Сетевой интерфейс имеет статическую пару ключей, назначенный порт и таблицу маршрутизации. В таблице маршрутизации для каждой записи хранится статический открытый ключ принимающей стороны.

Configuration 1a

Interface Public Key	**Interface Private Key**	**Listening UDP Port**
HIgo...8ykw	yAnz...fBmk	41414
Peer Public Key	**Allowed Source IPs**	
xTIB...p8Dg	10.192.122.3/32, 10.192.124.0/24	
TrMv...WXX0	10.192.122.4/32, 192.168.0.0/16	
gN65...z6EA	10.10.10.230/32	

Маршрутизация WireGuard

При отправке или получении пакета через интерфейс WireGuard из таблицы маршрутизации извлекается соответствующий открытый ключ, используемый для построения защищенного канала связи.

При получении пакета от других узлов их адреса добавляются в таблицу маршрутизации в поле Internet Endpoint. Данное поле может быть заполнено вручную на этапе конфигурации.

Пусть узел с конфигурацией 2b отправляет пакет узлу с предконфигурированном адресом 192.95.5.69:41414. Тогда на узле получателя при получении пакеты будет добавлена запись Internet Endpoint об узле отправителя (конфигурация 1b). Порты отправки и получения должны совпадать.

Маршрутизация WireGuard

Configuration 2b

Interface Public Key	**Interface Private Key**	**Listening UDP Port**
gN65...z6EA	gI6E...fWGE	21841
Peer Public Key	**Allowed Source IPs**	**Internet Endpoint**
HIgo...8ykw	0.0.0.0/0	192.95.5.69:41414

Configuration 1b

Interface Public Key	**Interface Private Key**	**Listening UDP Port**
HIgo...8ykw	yAnz...fBmk	41414
Peer Public Key	**Allowed Source IPs**	**Internet Endpoint**
xTIB...p8Dg	10.192.122.3/32, 10.192.124.0/24	
TrMv...WXX0	10.192.122.4/32, 192.168.0.0/16	
gN65...z6EA	10.10.10.230/32	192.95.5.64:21841

Процедура инкапсуляции WireGuard

Пакет с PT приходит на интерфейс WireGuard.

Определяется узел получателя и его открытый ключ.

Определяется ключ и счётчик для использования в ChaCha20Poly1305. Содержимое пакета шифруется.

К содержимому пакета добавляется заголовок.

Результат отправляется узлу в виде UDP пакета по адресу Internet Endpoint.

Криптография WireGuard

Основа – Noise «IK» (явная аутентичность ключа + forward secrecy)

IK(s, rs):

<- s

...

-> e, es, s, ss

<- e, ee, se

Первое передаваемое сообщение после handshake аутентифицирует инициатора.

Используемые примитивы – ECDH 25519, ChaCha20Poly1305, HKDF, BLAKE2s (MAC, PRF), BLAKE2s (Hash), HMAC

Использование метки времени

Благодаря 1-RTT нет необходимости хранить состояние при согласовании ключа.

Возможна атака повтором, в которой противник повторяет handshake, заставляя получателя разорвать существующую сессию (DOS).

Для защиты используется метка времени в формате TAI64N, используемая в теле первого передаваемого инициатором сообщения. Получатель хранит последнюю полученную метку времени. Все метки времени, младше полученной считаются недействительными.

Использование psk

Стойкость протокола основана на стойкости ECDH, которая сводится к сложности задачи дискретного логарифмирования.

При квантовом противники данная задача является эффективно вычислимой.

Возможна квантовая атака – противник хранит все текущие сессии, до момента получения в своё распоряжение квантового компьютера. Затем он может расшифровать весь перехваченный трафик, построив атаку на ECDH.

Возможно использование смягчения – использования статического симметричного ключа psk, который используется для «подмешивания» в ck symmetric state (см токен psk Noise) как начальный ключевой материал HKDF.

Cookie и DOS

Возможна атака на узел получателя в виде отправки множества сообщений в рамках handshake. Для ответов ему необходимо вычислять ECDH, что является трудозатратой операций и может привести к DOS.

Смягчение – при нагрузке узел может не отвечать на handshake а отправлять специальный ответ с cookie, ожидая особый ответ от инициатора.

Cookie и DOS

Получатель хранит секретное случайное значение, изменяемое каждые 2 минуты. Cookie – есть MAC от Ip инициатора, с использованием секретного значения в качестве ключа.

Инициатор при получении cookie использует его в качестве ключа для вычисления MAC от собственного запроса.

Получатель проверяет MAC от запроса пользователя.

Цель – привязка источника сообщений к его адресу. Количество сообщений с фиксированного Ip можно ограничивать стандартными средствами.

Cookie и DOS

Возможны проблемы – cookie не защищены, приходится отвечать на неаутентифицированные сообщения, противник может организовать DOS инициатора, отправляя невалидные cookie

Решение – больше cookie богу cookie

Использование двух различных mac

В качестве ответов на cookie

… приходится отвечать на неаутентифицированные сообщения

msg.mac1 – используем хэш от открытого ключа получателя в качестве ключа и вычисляем для всех байтов сообщения до msg.mac1. Присутствует в сообщении инициатора при handshake всегда, даже без нагрузки. Цель – узел должен знать с кем он общается (нет атак «из вне» от сторонних систем).

… cookie не защищены

Cookie будут шифроваться AEAD с использованием открытого ключа получателя. Цель – не знаешь PK получателя, не можешь пользоваться cookie.

… DOS на инициатора

Отвечая зашифрованной AEAD cookie инициатор добавляет в присоединённые данные (AD) msg.mac1 исходного сообщения инициатора. Цель – инициатор уверен, что cookie создана для его запроса.

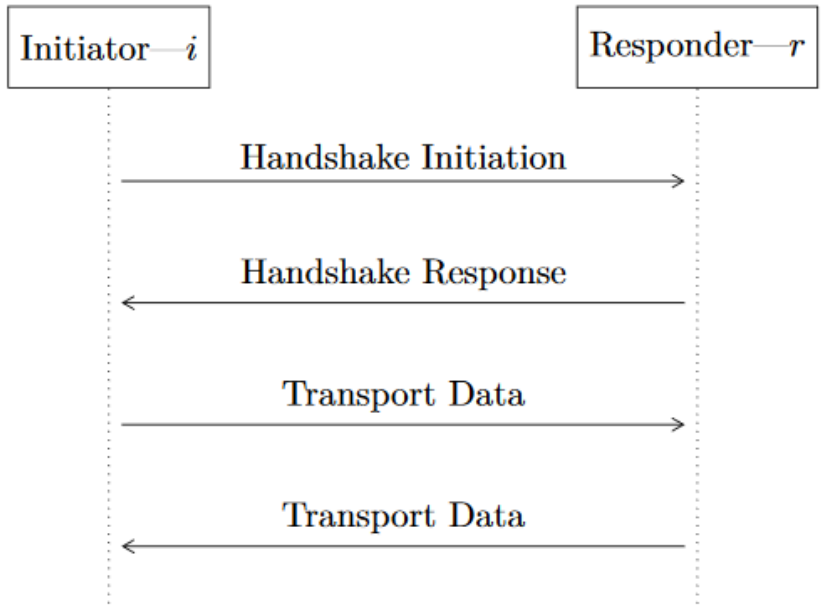
Второй MAC

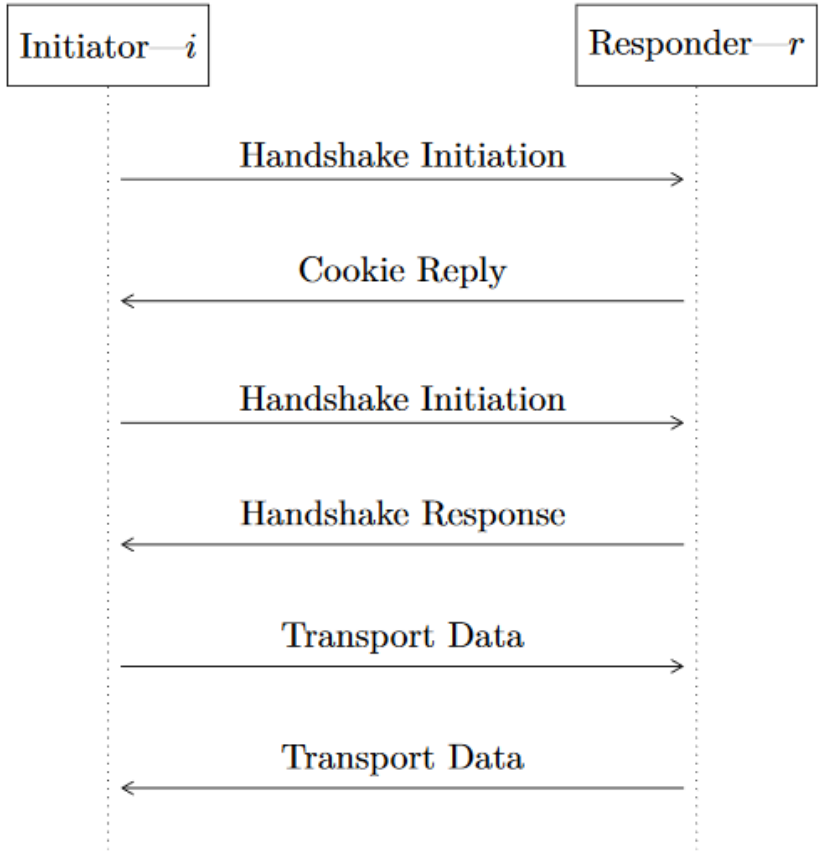
msg.mac2 использует cookie в качестве ключа и вычисляется для всех байтов сообщения до msg.mac2. Получатель проверяет данный MAC и отвечает на handshake. Используется только под нагрузкой.

Итого – получатель всегда проверяет msg.mac1 в первом сообщение, если под нагрузкой отправляет зашифрованную cookie с msg.mac1 в качестве AD, затем проверяет msg.mac2 и продолжает handshake. Если нагрузки нет - msg.mac2 не проверяется.

msg.mac1 и msg.mac2 аутентифицируют все передаваемые данные в сообщении на момент их формирования.

Итого


Initiator i
Responder r
Handshake Initiation
Handshake Response
Transport Data
Transport Data


Initiator i
Responder r
Handshake Initiation
Cookie Reply
Handshake Initiation
Handshake Response
Transport Data
Transport Data

Сообщение инициатора

type := 0x1 (1 byte)	reserved := 0^3 (3 bytes)
sender := I_i (4 bytes)	
ephemeral (32 bytes)	
static ($\widehat{32}$ bytes)	
timestamp ($\widehat{12}$ bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

$$C_i := \textsc{Hash}(\textsc{Construction})$$
$$H_i := \textsc{Hash}(C_i \parallel \textsc{Identifier})$$
$$H_i := \textsc{Hash}(H_i \parallel S_r^{pub})$$
$$(E_i^{priv}, E_i^{pub}) := \text{DH-}\textsc{Generate}()$$
$$C_i := \textsc{Kdf}_1(C_i, E_i^{pub})$$
$$\text{msg.ephemeral} := E_i^{pub}$$
$$H_i := \textsc{Hash}(H_i \parallel \text{msg.ephemeral})$$
$$(C_i, \kappa) := \textsc{Kdf}_2(C_i, \text{DH}(E_i^{priv}, S_r^{pub}))$$
$$\text{msg.static} := \textsc{Aead}(\kappa, 0, S_i^{pub}, H_i)$$
$$H_i := \textsc{Hash}(H_i \parallel \text{msg.static})$$
$$(C_i, \kappa) := \textsc{Kdf}_2(C_i, \text{DH}(S_i^{priv}, S_r^{pub}))$$
$$\text{msg.timestamp} := \textsc{Aead}(\kappa, 0, \textsc{Timestamp}(), H_i)$$
$$H_i := \textsc{Hash}(H_i \parallel \text{msg.timestamp})$$

$$\text{msg.mac1} := \textsc{Mac}(\textsc{Hash}(\textsc{Label-Mac1} \parallel S_{m'}^{pub}), \text{msg}_\alpha)$$
if $L_m = \epsilon$ or $\widetilde{L_m} \geq 120$:
$$\text{msg.mac2} := 0^{16}$$
otherwise:
$$\text{msg.mac2} := \textsc{Mac}(L_m, \text{msg}_\beta)$$

Сообщение получателя

type := 0x2 (1 byte)	reserved := 0^3 (3 bytes)
sender := I_r (4 bytes)	receiver := I_i (4 bytes)
ephemeral (32 bytes)	
empty ($\widehat{0}$ bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

$(E_r^{priv}, E_r^{pub}) := \text{DH-GENERATE}()$

$C_r := \text{KDF}_1(C_r, E_r^{pub})$

$\text{msg.ephemeral} := E_r^{pub}$

$H_r := \text{HASH}(H_r \parallel \text{msg.ephemeral})$

$C_r := \text{KDF}_1(C_r, \text{DH}(E_r^{priv}, E_i^{pub}))$

$C_r := \text{KDF}_1(C_r, \text{DH}(E_r^{priv}, S_i^{pub}))$

$(C_r, \tau, \kappa) := \text{KDF}_3(C_r, Q)$

$H_r := \text{HASH}(H_r \parallel \tau)$

$\text{msg.empty} := \text{AEAD}(\kappa, 0, \epsilon, H_r)$

$H_r := \text{HASH}(H_r \parallel \text{msg.empty})$

$\text{msg.mac1} := \text{MAC}(\text{HASH}(\text{LABEL-MAC1} \parallel S_{m'}^{pub}), \text{msg}_\alpha)$

if $L_m = \epsilon$ or $\widetilde{L_m} \geq 120$:

    $\text{msg.mac2} := 0^{16}$

otherwise:

    $\text{msg.mac2} := \text{MAC}(L_m, \text{msg}_\beta)$

Сообщение с cookie

type := 0x3 (1 byte)	reserved := 0^3 (3 bytes)
receiver := $I_{m'}$ (4 bytes)	
nonce := ρ^{24} (24 bytes)	
cookie ($\widehat{16}$ bytes)	

The secret variable, R_m, changes every two minutes to a random value, $A_{m'}$ represents a concatenation of the subscript's external IP source address and UDP source port, and M represents the msg.mac1 value of the message to which this is in reply. The remaining encrypted cookie reply field is populated as such:

$$\tau := \textsc{Mac}(R_m, A_{m'})$$

$$\text{msg.cookie} := \textsc{XAEAD}(\textsc{Hash}(\textsc{Label-Cookie} \parallel S_m^{pub}), \text{msg.nonce}, \tau, M)$$

Выработка транспортного ключа

- Нулевые начальные nonce, KDF от ck (см Noise).

$$(T_i^{send} = T_r^{recv}, T_i^{recv} = T_r^{send}) := \mathrm{KDF}_2(C_i = C_r, \epsilon)$$
$$N_i^{send} = N_r^{recv} = N_i^{recv} = N_r^{send} := 0$$
$$E_i^{priv} = E_i^{pub} = E_r^{priv} = E_r^{pub} = C_i = C_r := \epsilon$$

Использование транспортного ключа

type $\coloneqq$ 0x4 (1 byte)	reserved $\coloneqq 0^3$ (3 bytes)
receiver $\coloneqq I_{m'}$ (4 bytes)	
counter (8 bytes)	
packet ($\widehat{\|P\|}$ bytes)	

$$P \coloneqq P \parallel 0^{16 \cdot \lceil \|P\|/16 \rceil - \|P\|}$$

$$\text{msg.counter} \coloneqq N_m^{send}$$

$$\text{msg.packet} \coloneqq \text{AEAD}(T_m^{send}, N_m^{send}, P, \epsilon)$$

$$N_m^{send} \coloneqq N_m^{send} + 1$$

WireGuard

- Всё ещё в разработке. Текущая версия не рекомендована к использованию в продакшене.
- Активно внедряется с 2020 года
- Whitepaper https://www.wireguard.com/papers/wireguard.pdf