Cryptography in Real World protocols

Elena Kirshanova

Course “Information and Network Security”
Lecture 10
14 мая 2020 г.

We've looked at

I. Symmetric primitives:

- Pseudo random generators
- Stream ciphers
- Block ciphers
- MACs
- Hash functions
- Authenticated Encryption (AEAD)

II. Asymmetric primitives:

- Key Exchange
- Signature

The combination Key Exchange + Signature + AEAD rocks.

Part I

TLS

TLS: Transport Layer Security

TLS – protocol for establishing and maintaining a secure connection connection between a client and a server over the Internet.

I. SSL = Secure Socket Layer

- SSLv1 (1994) — unpublished
- SSLv2 (1995) — broken
- SSLv3 (1996) — supported

TLS: Transport Layer Security

TLS – protocol for establishing and maintaining a secure connection connection between a client and a server over the Internet.

I. SSL = Secure Socket Layer

- SSLv1 (1994) — unpublished
- SSLv2 (1995) — broken
- SSLv3 (1996) — supported

NOT TO BE USED

II. TLS = Transport Layer Security

- TLS 1.0 (1999) — RFC 2246
- TLS 1.1 (2006) — RFC 4346
- TLS 1.2 (2008) — RFC 5246
- TLS 1.3 (2018) — RFC 8448

IETF Standards

RFC = Request for Comments
IETF = Internet Engineering Task Force

High level structure of TLS

`Client` **Phase 1 Handshake** `Server`

choose primitives, params
authentication (at least server's)
common key derivation

High level structure of TLS

`Client` **Phase 1 Handshake** `Server`

choose primitives, params

authentication (at least server's)

common key derivation

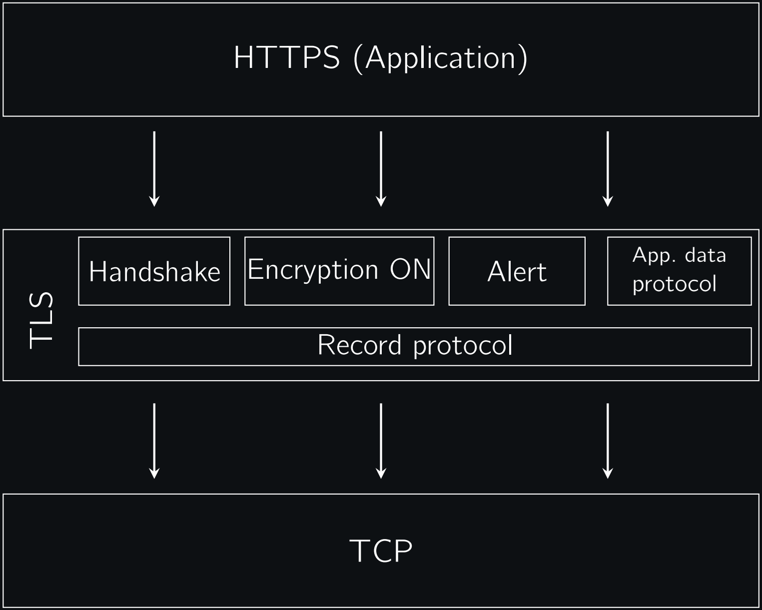
$$\downarrow k$$

Phase 2 TLS record protocol

AEAD to encrypt data under the key k

TLS lives in the TCP (transport layer), i.e., it assumes that packets arrive in order!

When TLS happens



TLS Handshake

Client Server

TLS Handshake

Client | **Server**

$\mathsf{pk}_c = g^a$, Nonce N_c, offer

$\longrightarrow$

offer: list of client's cipher suits

TLS Handshake

Client

Server

$\mathsf{pk}_c = g^a$, Nonce N_c, offer $\longrightarrow$

offer: list of client's cipher suits

1. chooses one cipher suit (Enc. scheme, hash)

TLS Handshake

Client

Server

$\text{pk}_c = g^a$, Nonce N_c, offer

→

offer: list of client's cipher suits

1. chooses one cipher suit (Enc. scheme, hash)
2. Computes $k_{\text{shared}} = g^{ab}$

k_{sh} - server enc. key

k_{sm} - server mac key

k_{ch} - client enc. key

k_{cm} - client mac key

$\text{pk}_s = g^b$, Nonce N_s, mode

←

mode: chosen cipher suits

TLS Handshake

Client

Server

$\mathsf{pk}_c = g^a$, Nonce N_c, offer →

offer: list of client's cipher suits

1. chooses one cipher suit (Enc. scheme, hash)
2. Computes $k_{\text{shared}} = g^{ab}$

k_{sh} - server enc. key

k_{sm} - server mac key

k_{ch} - client enc. key

k_{cm} - client mac key

← $\mathsf{pk}_s = g^b$, Nonce N_s, mode

mode: chosen cipher suits

Computes $k_{\text{shared}} = g^{ab}$

$k_{\text{sh}}, k_{\text{sm}}, k_{\text{ch}}, k_{\text{cm}}$

TLS Handshake

Client

Server

$\mathsf{pk}_c = g^a$, Nonce N_c, offer →

offer: list of client's cipher suits

1. chooses one cipher suit (Enc. scheme, hash)
2. Computes $k_{\text{shared}} = g^{ab}$

k_{sh} - server enc. key

k_{sm} - server mac key

k_{ch} - client enc. key

k_{cm} - client mac key

← $\mathsf{pk}_s = g^b$, Nonce N_s, mode

mode: chosen cipher suits

Computes $k_{\text{shared}} = g^{ab}$

$k_{\text{sh}}, k_{\text{sm}}, k_{\text{ch}}, k_{\text{cm}}$

←

$c_1 = \mathsf{Enc}(k_{\text{sh}},\ \text{Cert. request})$

$c_2 = \mathsf{Enc}(k_{\text{sh}},\ \text{Cert. Server})$

$c_3 = \mathsf{Enc}(k_{\text{sh}}, \mathsf{Sign}(\text{transcript}))$

$c_4 = \mathsf{Enc}(k_{\text{sh}}, \mathsf{MAC}(k_{\text{sm}}, \text{transcript}))$

TLS Handshake

Client — **Server**

Client → Server: $\mathsf{pk}_c = g^a$, Nonce N_c, offer

offer: list of client's cipher suits

Server:

1. chooses one cipher suit (Enc. scheme, hash)
2. Computes $k_{\text{shared}} = g^{ab}$

k_{sh} - server enc. key

k_{sm} - server mac key

k_{ch} - client enc. key

k_{cm} - client mac key

Server → Client: $\mathsf{pk}_s = g^b$, Nonce N_s, mode

mode: chosen cipher suits

Client: Computes $k_{\text{shared}} = g^{ab}$

$k_{\text{sh}}, k_{\text{sm}}, k_{\text{ch}}, k_{\text{cm}}$

Server → Client:

$c_1 = \text{Enc}(k_{\text{sh}}, \text{ Cert. request})$

$c_2 = \text{Enc}(k_{\text{sh}}, \text{ Cert. Server})$

$c_3 = \text{Enc}(k_{\text{sh}}, \text{Sign}(\text{transcript}))$

$c_4 = \text{Enc}(k_{\text{sh}}, \text{MAC}(k_{\text{sm}}, \text{transcript}))$

Client → Server:

$c_5 = \text{Enc}(k_{\text{ch}}, \text{ Cert. Client})$

$c_6 = \text{Enc}(k_{\text{ch}}, \text{Sign}(\text{transcript}))$

$c_7 = \text{Enc}(k_{\text{ch}}, \text{MAC}(k_{\text{cm}}, \text{transcript}))$

Client: $(k_{c \to s}, k_{s \to c}) = \mathcal{H}(\text{transcript})$

Server: $(k_{c \to s}, k_{s \to c}) = \mathcal{H}(\text{ transcript})$

TLS Record Layer

Data = $[m_1, \ldots, m_s]$

Client		Server
$k_{c\to s}$		$k_{c\to s}$
$k_{s\to c}$		$k_{s\to c}$

$$\underbrace{[\texttt{Meta data} || \ m_i \ || \ \texttt{Nonce}]}_{\text{AES-GCM-AEAD}(k_{c\to s})} \longrightarrow$$

Security and features

- Alert protocol is responsible for handling errors, warnings and session termination
- Security of TLS 1.3 is supported by strong analysis
- Update traffic keys feature: upon sending a `KeyUpdate` message Client and Server update $k_{c \to s}, k_{s \to c}$
- Pre-shared key handshake: more efficient Handshake Phase due to earlier sessions
- Forward secrecy: if an adversary compromises shared keys, the *previous* communication remains secure

Cipher Suites in TLS 1.3

Key Exchange	Certificates	Sym. encryption	Hash
ECDHE	ECDSA	AES_256_GCM	(H)SHA_384
DHE	RSA	CHACHA20_Poly1350	(H)SHA_256
RSA		AES_128_GCM	(H)SHA1
		AES_256_CBC	
		AES_128_CBC	
		3DES_CBC	

Cipher Suite Name Decoding

TLS 1.2

protocol name — authentication — MAC

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

key exchange — sym. encryption

- TLS → protocol name
- ECDHE → key exchange
- RSA → authentication
- AES_128_GCM → sym. encryption
- SHA256 → MAC

TLS 1.3 default suits:

TLS_AES_256_GCM_SHA384

TLS_CHACHA20_POLY1305_SHA256

TLS_AES_128_GCM_SHA256

Test your browser / server

Use

`https://www.ssllabs.com/index.html`

for a good SSL/TLS coverage

Or

`https://tls13.ulfheim.net/`

for an illustrated TLS Connection

Part II

Secure Messaging

Secure Messaging

A Secure Messaging (SM) allow two parties to communicate with each other with the following security conditions being satisfied:

Secure Messaging

A Secure Messaging (SM) allow two parties to communicate with each other with the following security conditions being satisfied:

- Correctness
- Privacy: attacker obtains no information about the messages sent unless a party is compromised
- Authenticity: the attacker cannot change, duplicate or inject messages
- Immediate decryption
- Message-loss resilience: if a message is lost, communication continues
- Forward secrecy: all messages exchanged *before* a compromise remain hidden to an attacker
- Post-compromise security: the parties can recover *after* a compromise

Secure Messaging protocol: Signal

The Signal Protocol, designed by Open Whisper Systems, is an example of Secure Messaging.

- deployed in many apps like WhatsApp, Facebook Messenger, Skype
- every message is encrypted and authenticated using a fresh symmetric key
- satisfies the above security conditions

Description of Signal:
`https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf`

It's analysis: `https://eprint.iacr.org/2018/1037.pdf`

Primitives for SM Protocols

Correctness
Privacy
Authenticity
Immediate decryption
Message-loss resilience
Forward security

} AEAD (symmetric primitive)

AEAD – Authenticated Encryption with Associated Data

Post-compromise security } CKA (asymmetric primitive)

CKA – Continuous Key Agreement

Signal: high level symmetric part

A		B
`Sender`	$\leftarrow k_{\text{shared}} \rightarrow$	`Receiver`

Signal: high level symmetric part

A		**B**
Sender	$\leftarrow k_{\text{shared}} \rightarrow$	**Receiver**
$s_{A,0} \leftarrow \texttt{InitSender}(k)$		$s_{B,0} \leftarrow \texttt{InitReceiver}(k)$

Signal: high level symmetric part

A		**B**
Sender	$\leftarrow k_{\text{shared}} \rightarrow$	**Receiver**
$s_{\mathsf{A},0} \leftarrow \texttt{InitSender}(k)$		$s_{\mathsf{B},0} \leftarrow \texttt{InitReceiver}(k)$
$s_{\mathsf{A},1}, c_1 \leftarrow \texttt{Send}(m_1)$	$\xrightarrow{c_1}$	
$s_{\mathsf{A},2}, c_2 \leftarrow \texttt{Send}(m_2)$	$\xrightarrow{c_2}$	
		$s_{\mathsf{B},1}, m_1 \leftarrow \texttt{Receive}(c_1)$
		$s_{\mathsf{B},2}, m_2 \leftarrow \texttt{Receive}(c_2)$
	$\vdots$	

Signal: high level symmetric part

A Sender		B Receiver
	$\leftarrow k_{\text{shared}} \rightarrow$	
$s_{A,0} \leftarrow \texttt{InitSender}(k)$		$s_{B,0} \leftarrow \texttt{InitReceiver}(k)$
$s_{A,1}, c_1 \leftarrow \texttt{Send}(m_1)$	$\xrightarrow{c_1}$	
$s_{A,2}, c_2 \leftarrow \texttt{Send}(m_2)$	$\xrightarrow{c_2}$	
		$s_{B,1}, m_1 \leftarrow \texttt{Receive}(c_1)$
		$s_{B,2}, m_2 \leftarrow \texttt{Receive}(c_2)$
	$\vdots$	

- think about **Send** as of encryption, **Receive** as of decryption
- $s_{A,i}$ – A's $i-$th state
- $s_{B,i}$ – B's $i-$th state
- the states should remain secure
- ciphertexts c_i may not come to B in order!

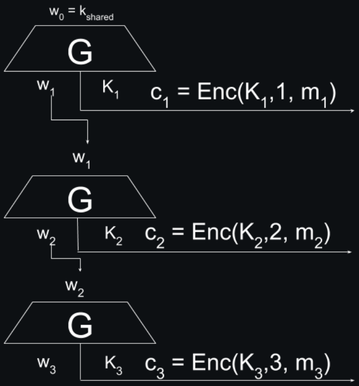
Signal: AEAD + PRG

Let Enc(), Dec() be encryption/decryption procedures of AEAD (see Lec. 7)

$G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ – cryptographic pseudo-random generator (see Lec. 2)

A

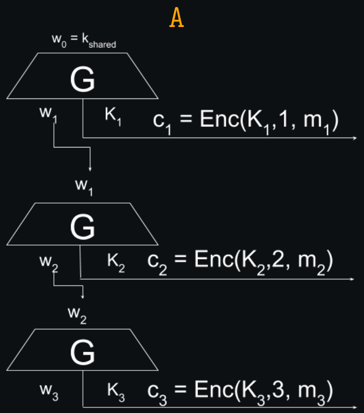
B



Signal: AEAD + PRG

Let Enc(), Dec() be encryption/decryption procedures of AEAD (see Lec. 7)

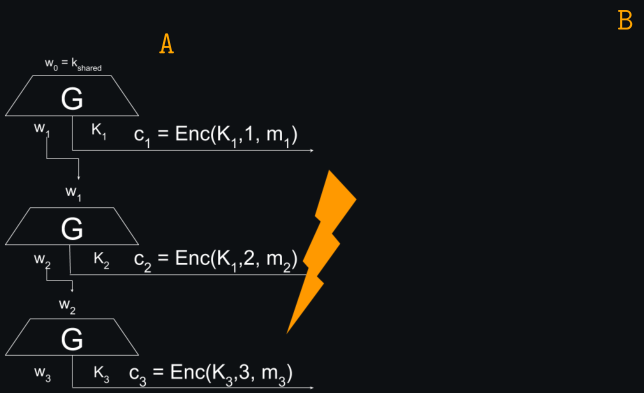
G : $\{0,1\}^n \rightarrow \{0,1\}^{2n}$ – cryptographic pseudo-random generator (see Lec. 2)



Signal: AEAD + PRG

Let Enc(), Dec() be encryption/decryption procedures of AEAD (see Lec. 7)

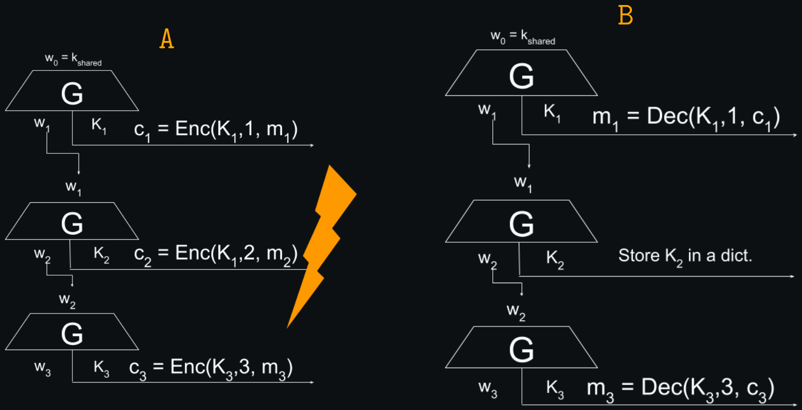
$G : \{0,1\}^n \to \{0,1\}^{2n}$ – cryptographic pseudo-random generator (see Lec. 2)



Signal: AEAD + PRG

Let Enc(), Dec() be encryption/decryption procedures of AEAD (see Lec. 7)

$G : \{0,1\}^n \to \{0,1\}^{2n}$ – cryptographic pseudo-random generator (see Lec. 2)



All w_i are erased when no further needed.

Signal: asymmetric part

How to get k_{shared}?

Signal: asymmetric part

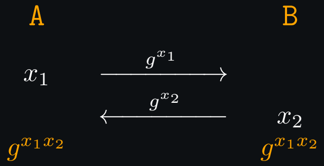
How to get k_{shared}? You know the answer: use Key Exchange!

Signal is using Continuous Key Exchange based on Diffie-Hellman.

Signal: asymmetric part

How to get k_{shared}? You know the answer: use Key Exchange!

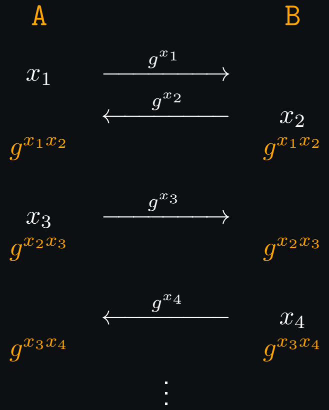
Signal is using Continuous Key Exchange based on Diffie-Hellman.



Signal: asymmetric part

How to get k_{shared}? You know the answer: use Key Exchange!

Signal is using Continuous Key Exchange based on Diffie-Hellman.



- At time i the shared key is $g^{x_i x_{i-1}}$
- A shared key is generated each time a party switches from `Receiver` to `Sender`
- If at some point $g^{x_i x_{i-1}}$ is compromised (attacker knows x_i), the parties recover privacy within two rounds.

The last slide

This is the end of the lectures!

If you want to work on crypto, here is the list of potential projects/thesis topics:
`https://crypto-kantiana.com/thesis_topics.html`

The last slide

This is the end of the lectures!

If you want to work on crypto, here is the list of potential projects/thesis topics:
`https://crypto-kantiana.com/thesis_topics.html`

Stay healthy and hope to see you soon!