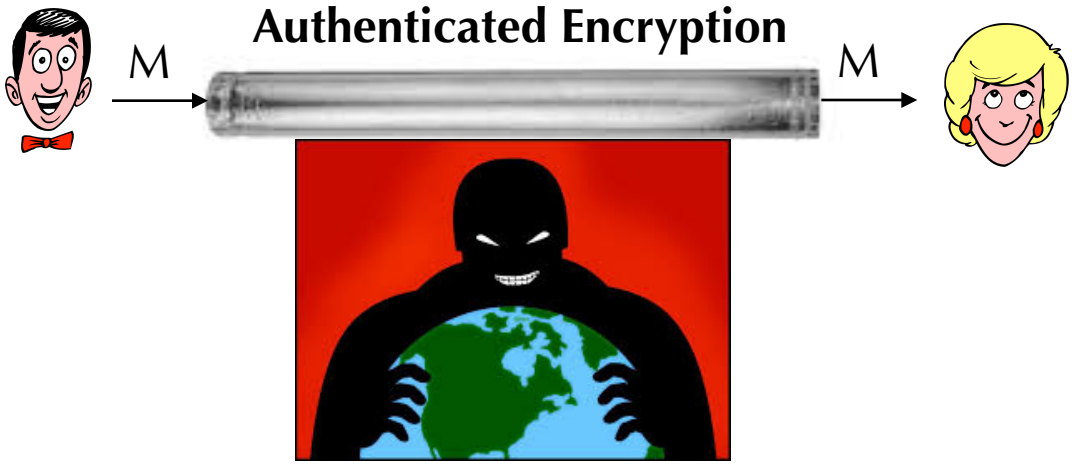
(Partially Specified) Secure Channels

Tom Shrimpton
University of Florida

Prologue: Review of AE



Prologue: Review of AE

Probabilistic or deterministic AE?

Nonce based AE?

What happens if a nonce repeats?

Do I need to support associated data?

What primitives should we build upon?

encryption + MAC? (tweakable) wide-block cipher? sponges? ...

What should happen when decryption fails?

Is it safe to provide multiple, descriptive exceptions/error messages?

Stop all future processing, or just for this message?

What kind of information can decryption safely leak?

Safe to release plaintext data "early"?

Online encryption/decryption property?

"Atomic" plaintexts/ciphertexts, or stream-based?

(Authenticated encryption != Secure Channel)

Prologue: Review of AE

A (probabilistic) authenticated encryption Scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a triple of algorithms

Key-generation algorithm $\mathcal{K}$ samples from a specified key space

Encryption algorithm $\mathcal{E} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ $\quad C \xleftarrow{\$} \mathcal{E}_K(M)$

Decryption algorithm $\mathcal{D} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ $\quad M \leftarrow \mathcal{D}_K(C)$

Always deterministic

Correctness:

$$\forall K \in \mathcal{K}, \forall M \in \{0,1\}^* : \Pr\left[C \xleftarrow{\$} \mathcal{E}_K(M) : C = \perp \text{ or } \mathcal{D}_K(C) = M \right] = 1$$

(note: logically equivalent to $C \neq \perp \implies \mathcal{D}_K(C) = M$)

Prologue: Review of AE

Privacy: Indistinguishability from random bits (IND$-CPA)

$\underline{\mathbf{Exp}_{\Pi}^{\text{ind\$-cpa}}(A)}$:
$K \xleftarrow{\$} \mathcal{K}$
$d \xleftarrow{\$} \{0,1\}$
$d' \xleftarrow{\$} A^{\mathcal{O}(\cdot)}$
If $d' = d$ then Return 1
Return 0

$\underline{\textbf{Oracle } \mathcal{O}(M)}$
$Y_1 \xleftarrow{\$} \mathcal{E}_K(M)$
$Y_0 \xleftarrow{\$} \{0,1\}^{|Y_1|}$
Return Y_d

$$\mathbf{Adv}_{\Pi}^{\text{ind\$-cpa}}(A) = 2\Pr\left[\mathbf{Exp}_{\Pi}^{\text{ind\$-cpa}}(A) = 1\right] - 1$$

Prologue: Review of AE

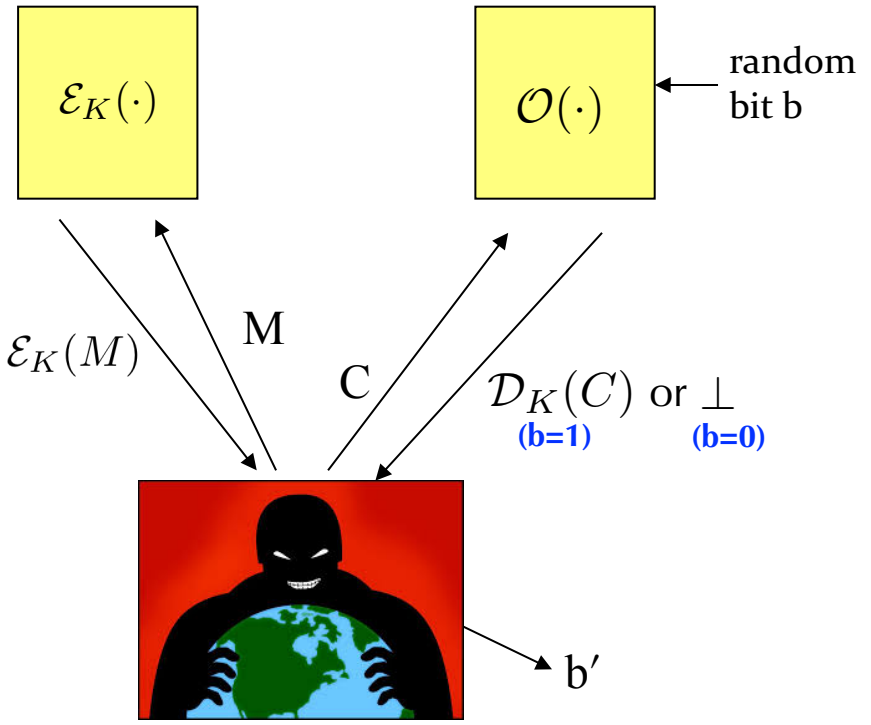
Authenticity: Integrity of Ciphertexts (INT-CTXT)

(Bellare, Rogaway AC' 00) (Katz, Yung FSE' 00) (Bellare, Namprempre AC'00)

$\underline{\mathbf{Exp}^{\text{int-ctxt}}_{\Pi}(A)}$:
$K \xleftarrow{\$} \mathcal{K}$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} A^{\mathcal{E}_K(\cdot),\mathcal{O}(\cdot)}$
If $b' = b$ then Return 1
Return 0

$\underline{\text{Oracle } \mathcal{O}(C)}$:
If $b = 0$ then Return $\perp$
Return $\mathcal{D}_K(C)$

$$\mathbf{Adv}^{\text{int-ctxt}}_{\Pi}(A) = 2\Pr(\mathbf{Exp}^{\text{int-ctxt}}_{\Pi}(A) = 1) - 1$$



To prevent "trivial wins" of the game, adversary is forbidden to ask C of the right oracle if C was returned by the left oracle

Prologue: Review of AE

Working definition of "AE secure":

If encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is
IND$-CPA secure **and** INT-CTXT secure then it is "AE secure".

ctxts look like
random bitstrings

dishonestly created ctxts
decrypt to $\perp$

Prologue: Review of AE

AE via "generic composition" of encryption and MAC

SSH: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M)||F_{K_2}(M)$ **"Encrypt and MAC"**

SSL/TLS: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M||F_{K_2}(M))$ **"MAC then Encrypt"**

IPSec: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M)||F_{K_2}(\overline{\mathcal{E}}_{K_1}(M))$ **"Encrypt then MAC"**

(Bellare, Namprempre AC' 00)


"Which of EaM, MtE, EtM gives a secure AE scheme, given a secure encryption scheme and a secure MAC?"

Prologue: Review of AE

AE via "generic composition" of encryption and MAC

SSH: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M)||F_{K_2}(M)$ **"Encrypt and MAC"**

SSL/TLS: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M||F_{K_2}(M))$ **"MAC then Encrypt"**

IPSec: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M)||F_{K_2}(\overline{\mathcal{E}}_{K_1}(M))$ **"Encrypt then MAC"**

(Bellare, Namprempre AC' 00)

"Which of EaM, MtE, EtM gives a secure AE scheme, given a secure encryption scheme and a secure MAC?"

EtM

Prologue: Review of AE

AE via "generic composition" of encryption and MAC

SSH: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M)||F_{K_2}(M)$ **"Encrypt and MAC"**

SSL/TLS: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M||F_{K_2}(M))$ **"MAC then Encrypt"**

IPSec: $\mathcal{E}_{K_1,K_2}(M) = \overline{\mathcal{E}}_{K_1}(M)||F_{K_2}(\overline{\mathcal{E}}_{K_1}(M))$ **"Encrypt then MAC"**

(Bellare, Namprempre AC' 00)


"Which of EaM, MtE, EtM gives a secure probabilistic AE scheme, given a secure probabilistic encryption scheme and a secure MAC?"

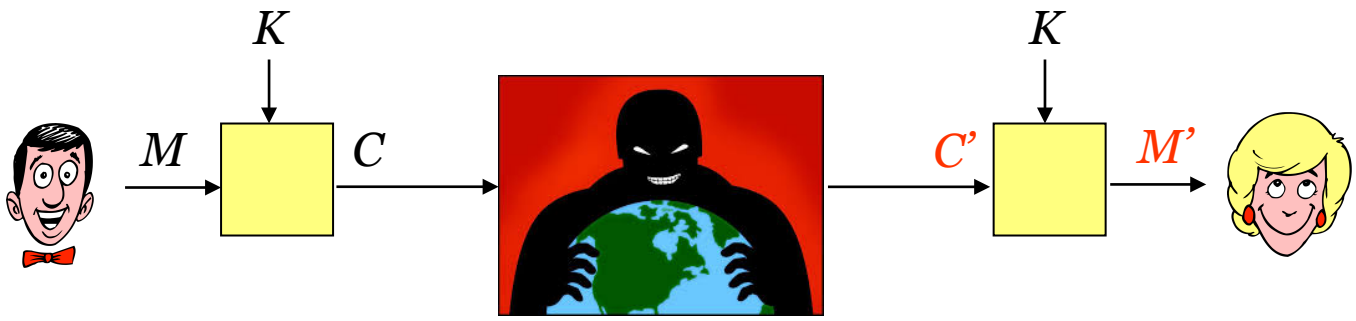
EtM

(We will come back to the more modern viewpoint on AE(AD) in a bit…)

What is a “secure channel”?

What is a "secure channel"?

Intuitively, a **secure channel** should provide

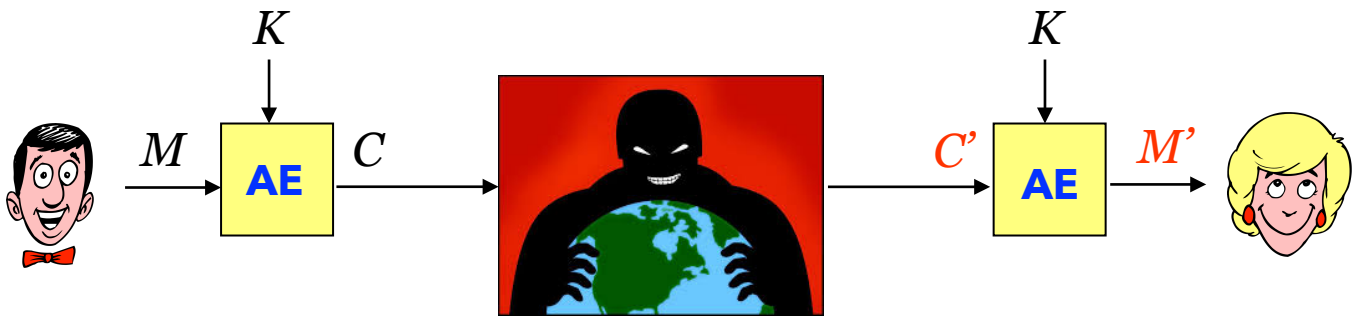


Privacy for the message *M*

Integrity protection for the transmission

What is a "secure channel"?

Intuitively, a **secure channel** should provide



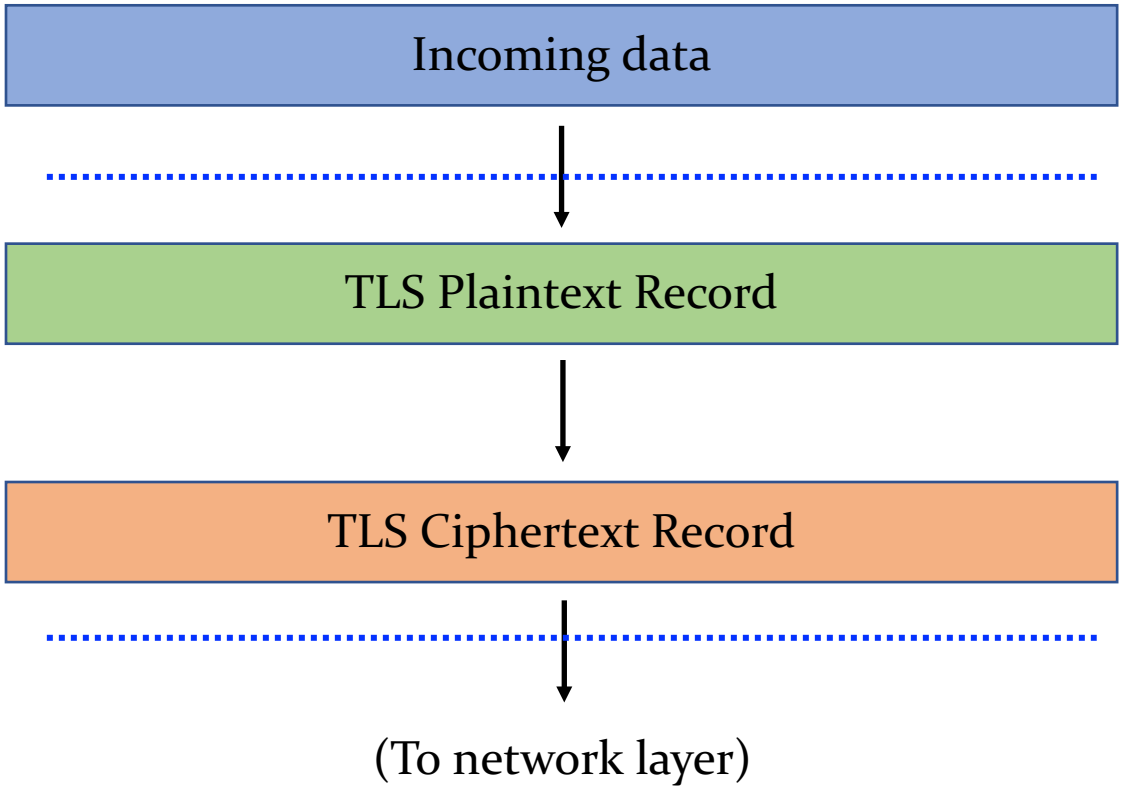
Privacy for the message *M*
(IND$-CPA)

Integrity protection for the transmission
(INT-CTXT)

This sounds a lot like **authenticated encryption**!

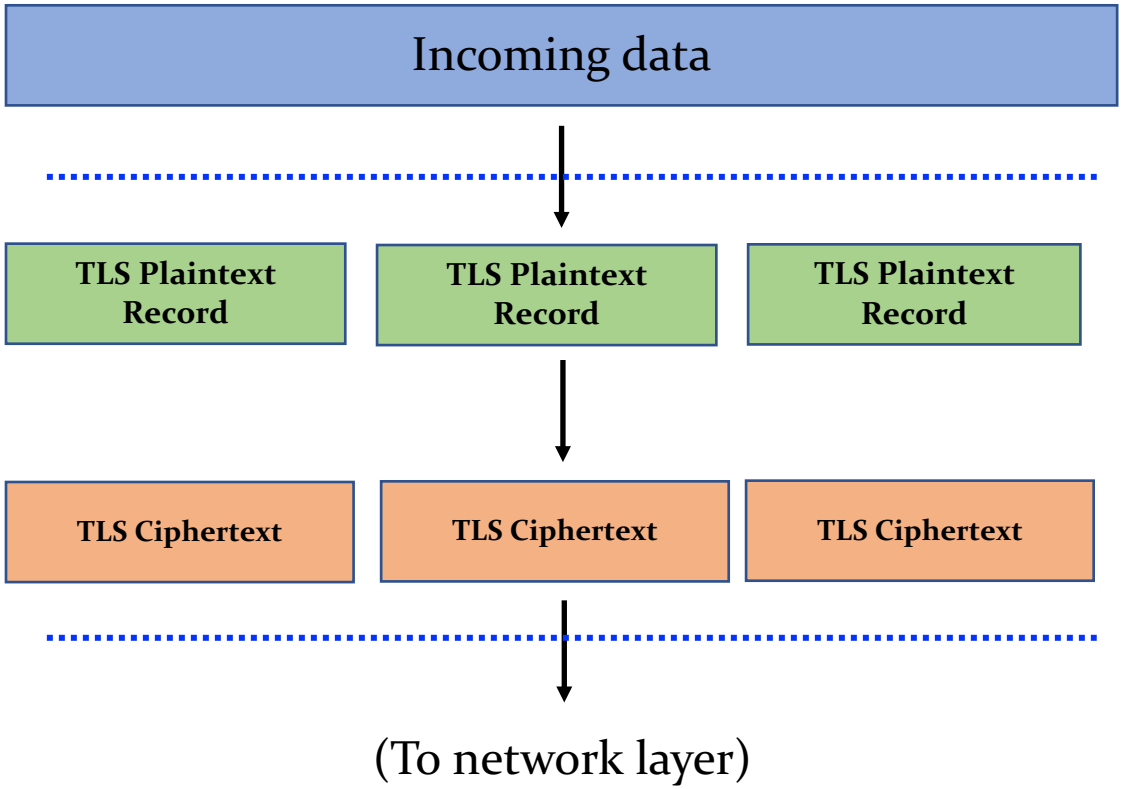
Real-world secure channels

Let's look at a **real** secure channel specification: **TLS v 1.2**



Real-world secure channels

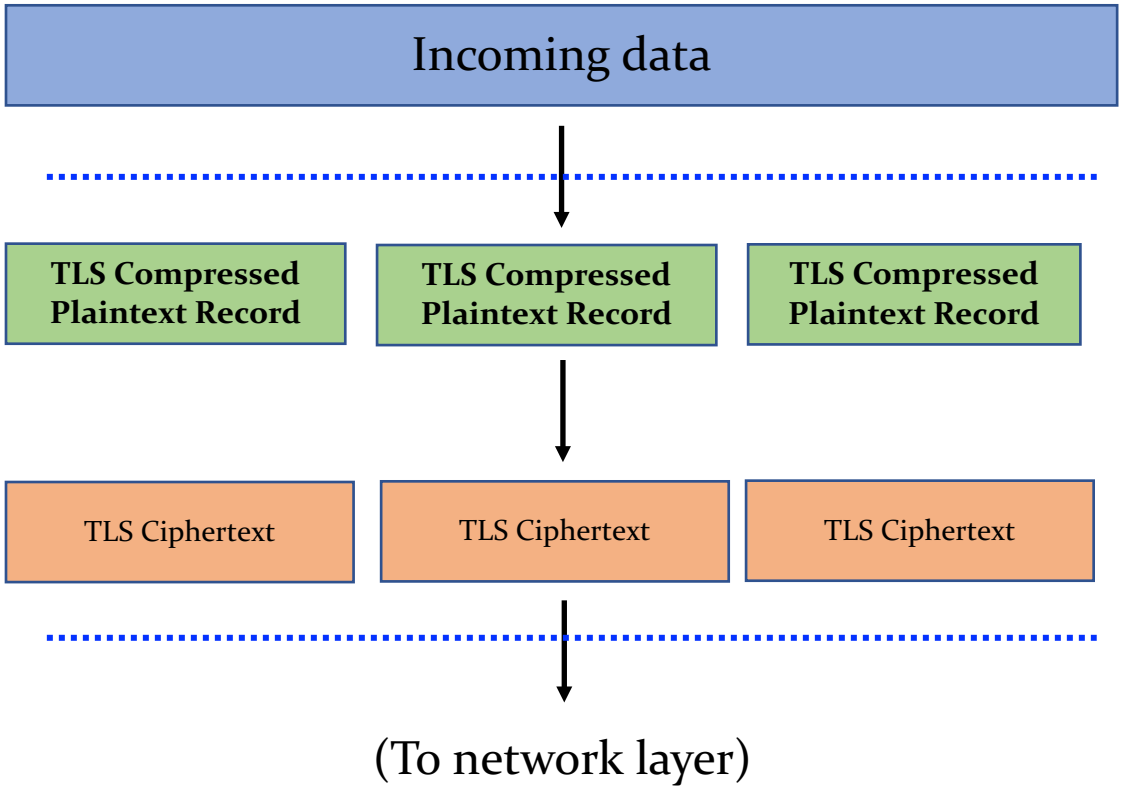
Let's look at a **real** secure channel: TLS 1.2 record layer



"The record layer fragments information blocks into TLSPlaintext records carrying data in chunks of 2^14 bytes or less."

Real-world secure channels

Let's look at a **real** secure channel: TLS 1.2 record layer



"All records are compressed using the compression algorithm defined in the current session state..."

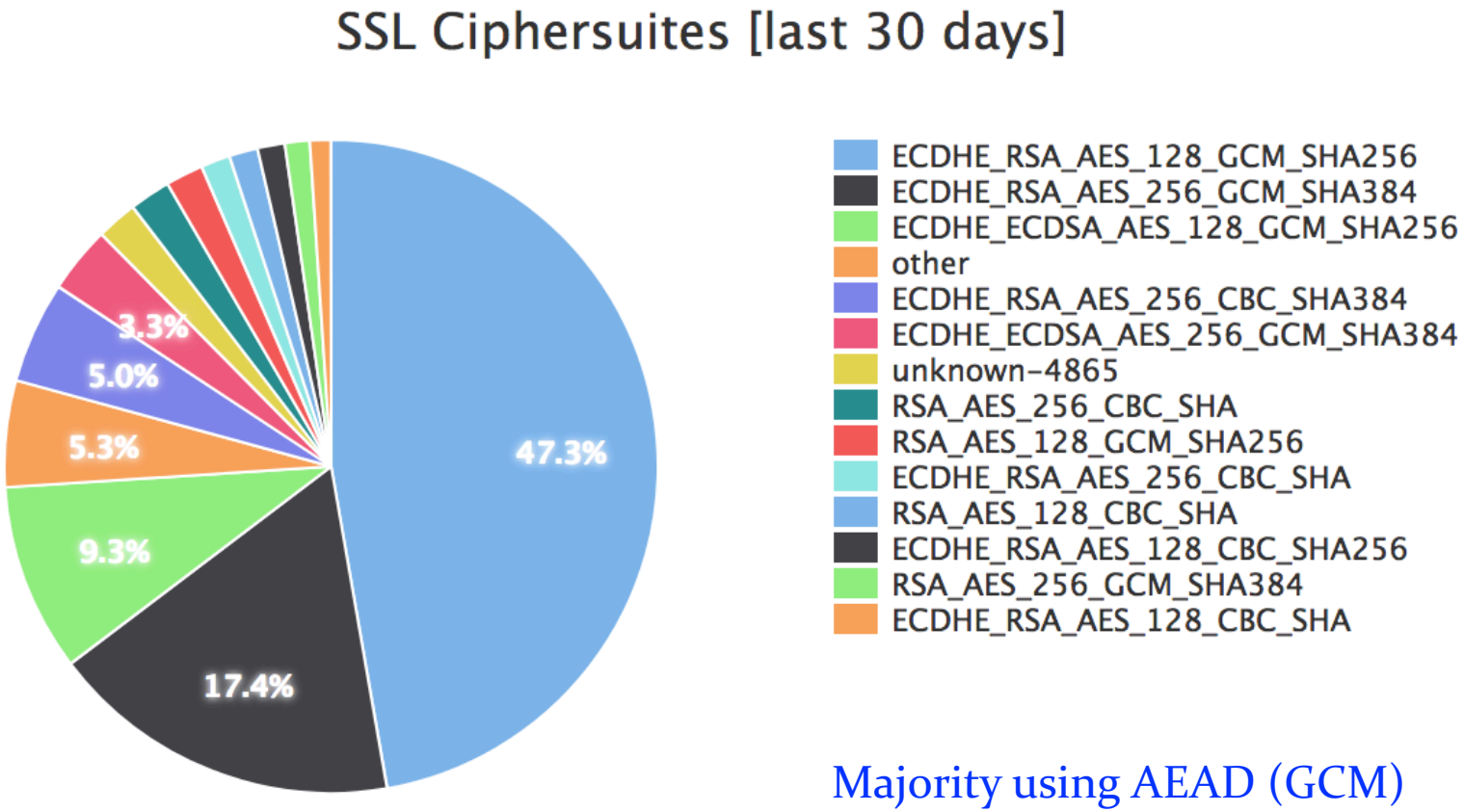
(Usually, the compression algorithm is "no compression")

Observations

1. The application may pass in arbitrary length data, but an RFC-compliant implementation of **the TLS secure channel may fragment however it likes**, so long as fragments are at most 2^14 bytes long.

Distribution of SSL/TLS Ciphersuites

Just how does TLS transform (compressed) plaintext records into ciphertexts?

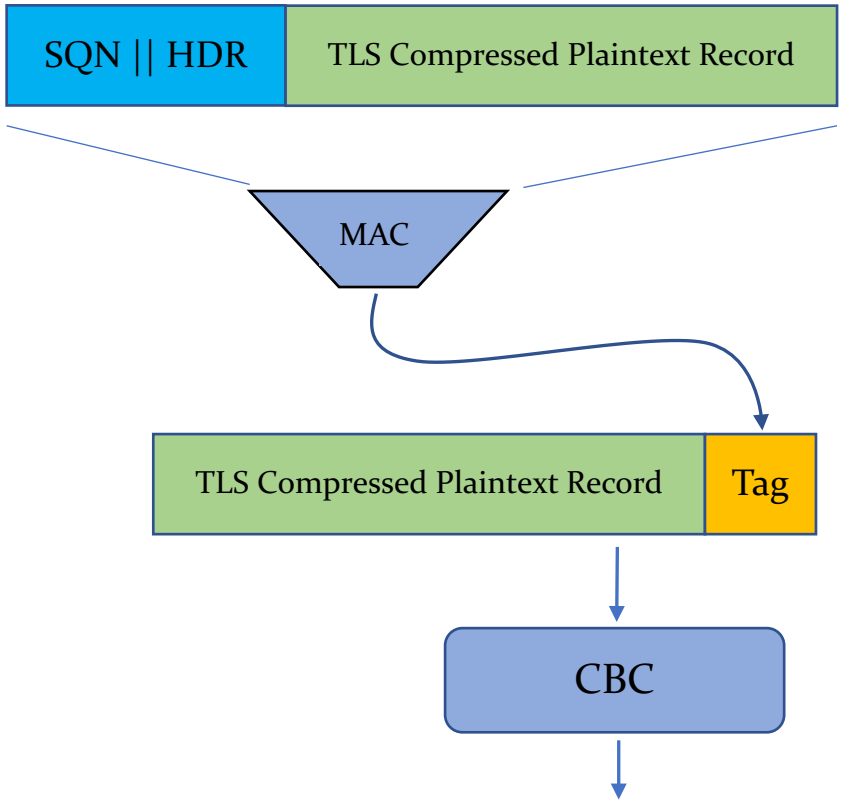


Majority using AEAD (GCM)

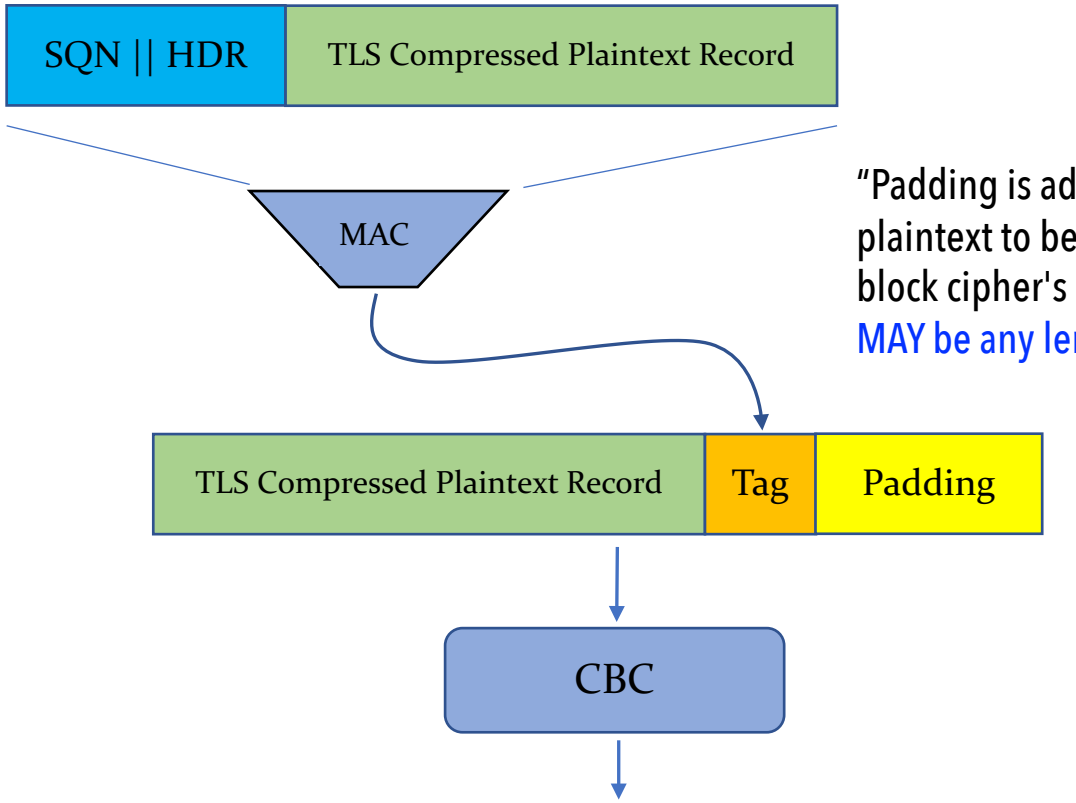
About 12.5% using CBC mode

TLS v1.2 CBC Encryption

"The encryption and MAC functions translate [plaintext data] into a TLSCiphertext… The MAC of the [plaintext data] also includes a sequence number so that missing, extra, or repeated messages are detectable."

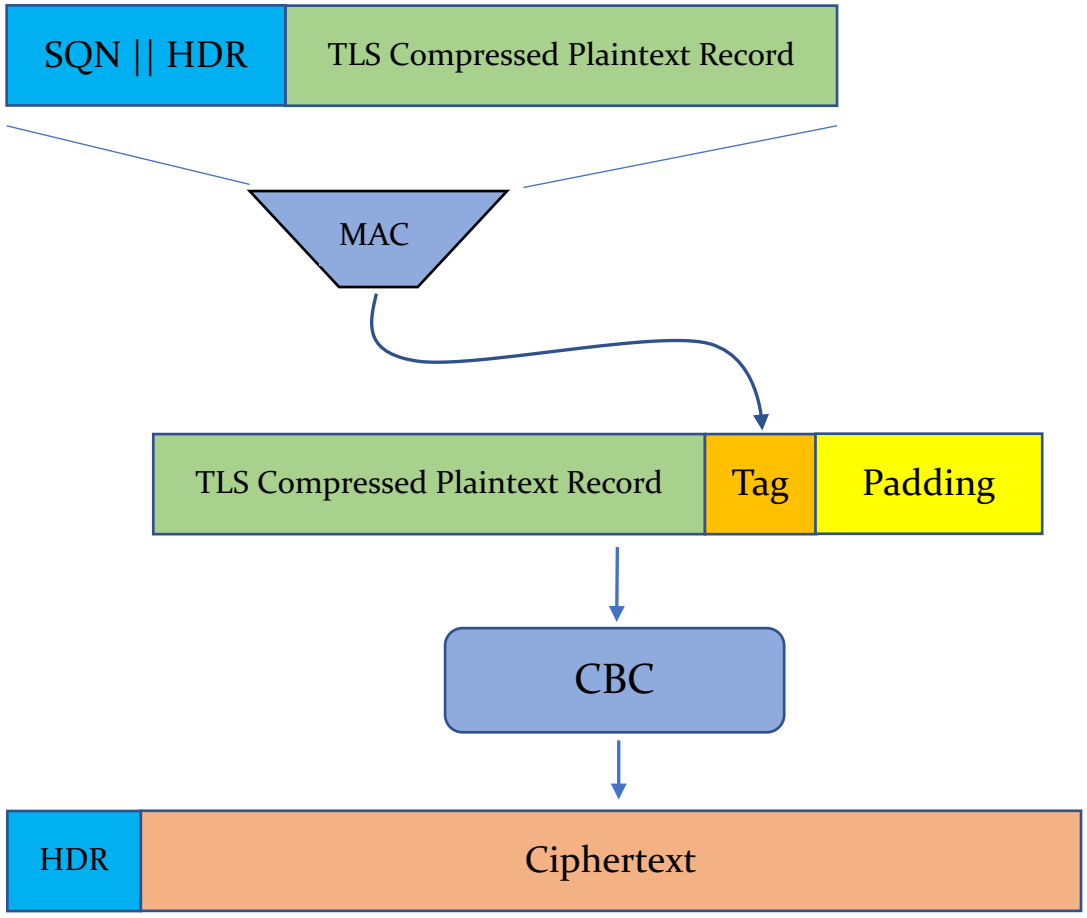


TLS v1.2 CBC Encryption



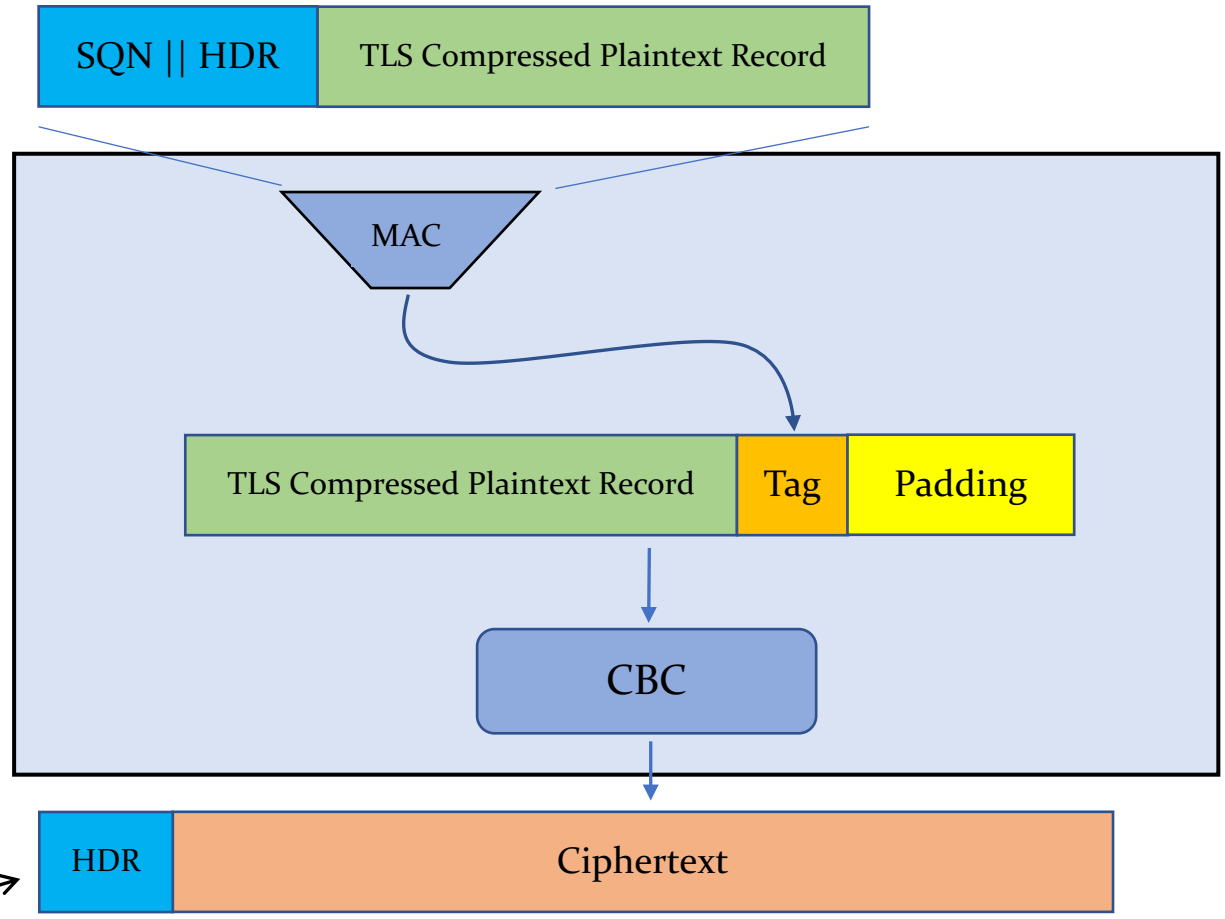
"Padding is added to force the length of the plaintext to be an integral multiple of the block cipher's block length. The padding MAY be any length up to 255 bytes"

TLS v1.2 CBC Encryption



Note: SQN **not** sent

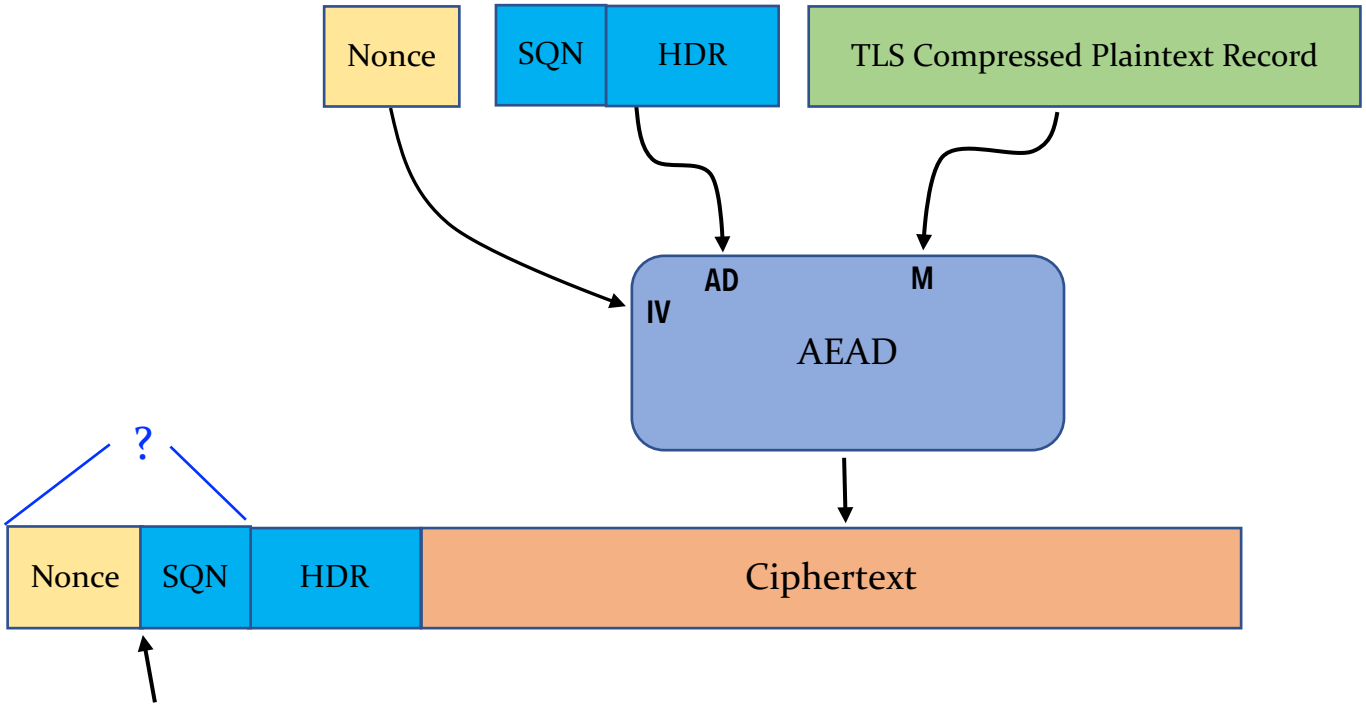
TLS v1.2 CBC Encryption



Here's the underlying AE scheme: MAC-then-Encode-then-Encrypt

Note: SQN **not** sent

TLS v1.2 AEAD Encryption



Whether or not these are sent,
or derived from sender/receiver-side
state is left to the implementation

Interlude: Reconsidering Generic Composition

(Bellare, Namprempre AC' 00)


"Which of EaM, EtM, MtE gives a secure probabilistic AE scheme, given a secure probabilistic encryption scheme and a secure MAC?"

EtM

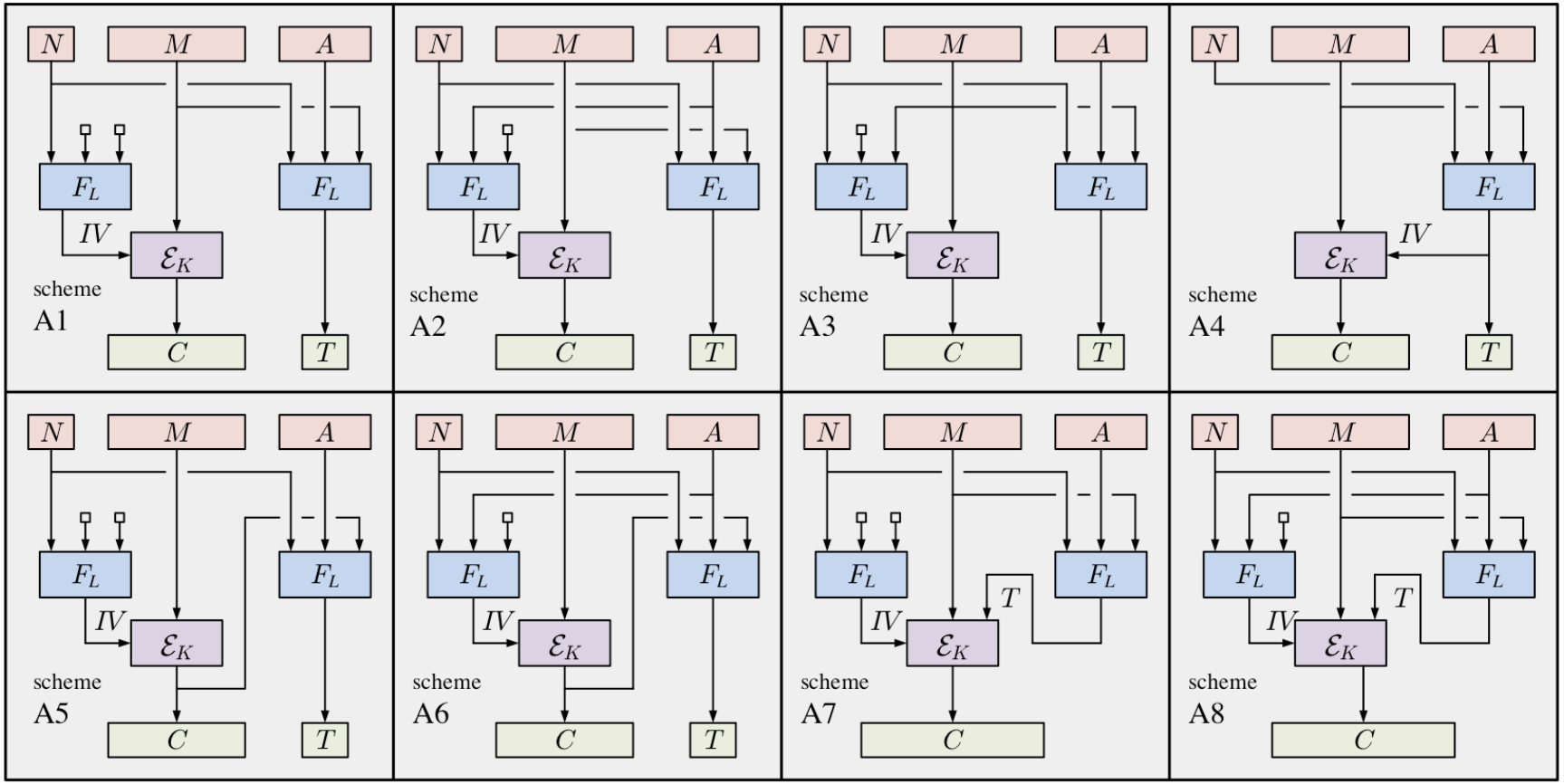
15 years!

(Namprempre, Rogaway, S. EC' 14)

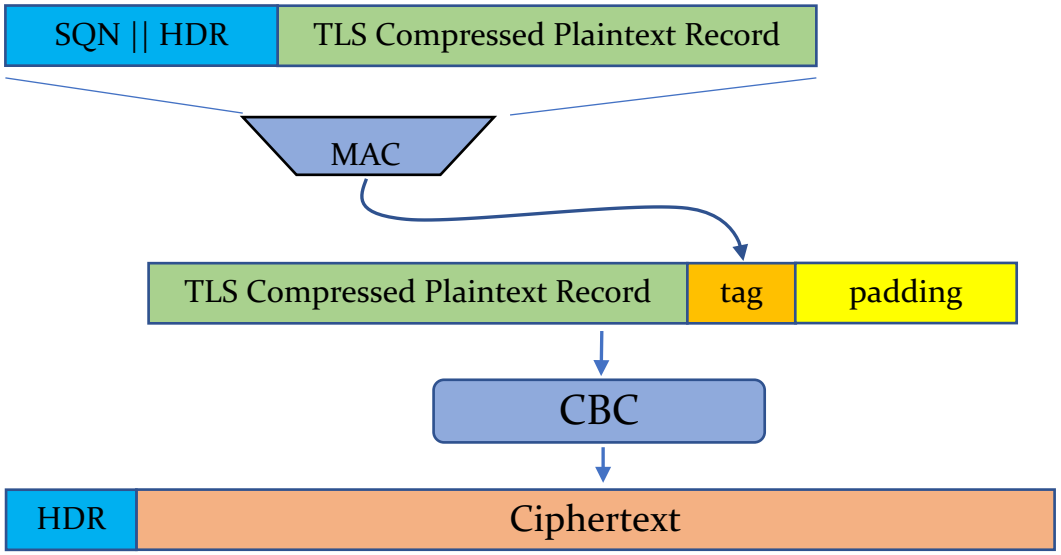


"What are **all** of ways to build an IV-based AEAD scheme that is **secure with nonce IVs**, from a secure IV-based encryption scheme and a secure PRF?"

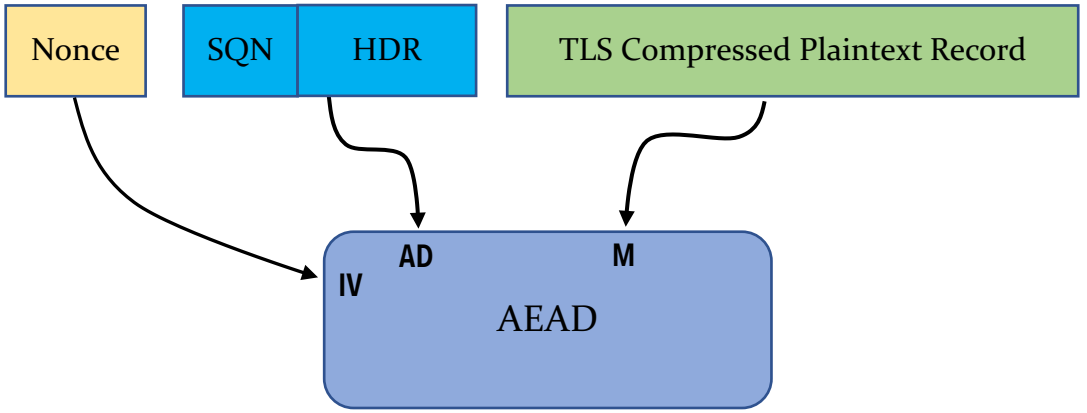
Interlude: Reconsidering Generic Composition



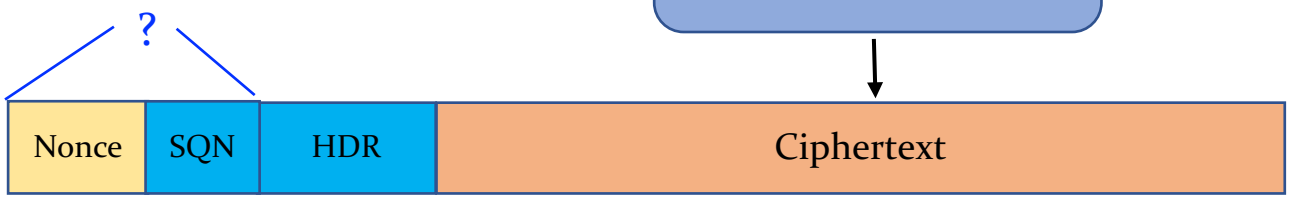
Back to the main story…



SQN not sent



SQN may not sent



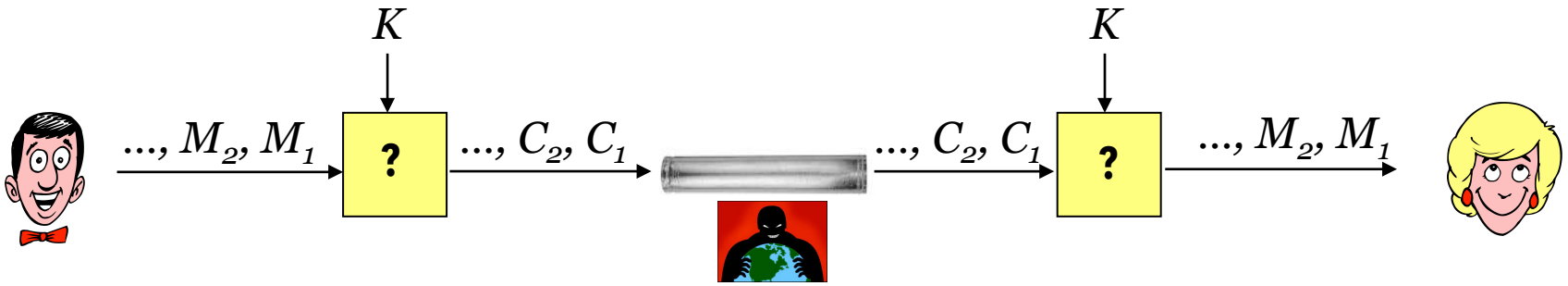
Observations

1. The application may pass in arbitrary length data, but an RFC-compliant implementation of the TLS secure channel may fragment however it likes, so long as fragments are at most 2^14 bytes long.

2. TLS records are embellished with a sequence number (SQN), **to protect against out-of-order delivery** (includes replay, fragment dropping, etc.)

If encryption is CBC mode, **the SQN is not sent**.
If encryption is AEAD **the SQN** may or **may not be sent**.

AE != secure channel



Apparently, a secure channel should provide:

Privacy for messages	✓ AE
Integrity protection for the transmission	✓ AE
Protection against out-of-order delivery	✗ AE

If the SQN is not sent, sender **and receiver** need to be **stateful** for proper modeling.

Authenticity when decryption is stateful

[Bellare, Kohno, Namprempre '04] **First to capture security against replay, out of order delivery, etc.**

Experiment $\mathbf{Exp}^{\text{int-sfctxt}}_{\mathcal{SE}}(A_{\text{sfctxt}})$
$K \xleftarrow{R} \mathcal{K}$
$i \leftarrow 0$; $j \leftarrow 0$; $\mathsf{phase} \leftarrow 0$
Run $A_{\text{ctxt}}^{\mathcal{E}_K(\cdot),\mathcal{D}^*_K(\cdot)}$
Reply to $\mathcal{E}_K(M)$ queries as follows:
$i \leftarrow i+1$; $C_i \xleftarrow{R} \mathcal{E}_K(M_b)$
$A_{\text{sfctxt}} \Leftarrow C_i$
Reply to $\mathcal{D}^*_K(C)$ queries as follows:
$j \leftarrow j+1$; $M \leftarrow \mathcal{D}_K(C)$
If $j > i$ or $C \neq C_j$ then $\mathsf{phase} \leftarrow 1$
If $M \neq \perp$ and $\mathsf{phase} = 1$ then return 1
If $M \neq \perp$ then $A_{\text{sfctxt}} \Leftarrow 1$
Else $A_{\text{sfctxt}} \Leftarrow 0$
Until A_{sfctxt} halts
Return 0

Channel is initially "in sync"

Sequencing of sent ciphertexts

Sequencing of received ciphertexts

Authenticity when decryption is stateful

[Bellare, Kohno, Namprempre '03]: **First to capture security against replay, out of order delivery, etc.**

Experiment $\mathbf{Exp}^{\text{int-sfctxt}}_{\mathcal{SE}}(A_{\text{sfctxt}})$
$K \xleftarrow{R} \mathcal{K}$
$i \leftarrow 0$; $j \leftarrow 0$; phase $\leftarrow 0$
Run $A^{\mathcal{E}_K(\cdot),\mathcal{D}^*_K(\cdot)}_{\text{ctxt}}$
Reply to $\mathcal{E}_K(M)$ queries as follows:
$i \leftarrow i+1$; $C_i \xleftarrow{R} \mathcal{E}_K(M_b)$
$A_{\text{sfctxt}} \Leftarrow C_i$
Reply to $\mathcal{D}^*_K(C)$ queries as follows:
$j \leftarrow j+1$; $M \leftarrow \mathcal{D}_K(C)$
If $j > i$ or $C \neq C_j$ then phase $\leftarrow 1$
If $M \neq \perp$ and phase $= 1$ then return 1
If $M \neq \perp$ then $A_{\text{sfctxt}} \Leftarrow 1$
Else $A_{\text{sfctxt}} \Leftarrow 0$
Until A_{sfctxt} halts
Return 0

Channel is initially "in sync"

Sequencing of sent ciphertexts

Sequencing of received ciphertexts

Channel is "out of sync", relative to what has been sent.
GAME ON!

Authenticity when decryption is stateful

[Bellare, Kohno, Namprempre '03]: **First to capture security against replay, out of order delivery, etc.**

Experiment $\mathbf{Exp}^{\text{int-sfctxt}}_{\mathcal{SE}}(A_{\text{sfctxt}})$
$K \xleftarrow{R} \mathcal{K}$
$i \leftarrow 0$; $j \leftarrow 0$; phase $\leftarrow 0$
Run $A^{\mathcal{E}_K(\cdot),\mathcal{D}^*_K(\cdot)}_{\text{ctxt}}$
Reply to $\mathcal{E}_K(M)$ queries as follows:
$i \leftarrow i+1$; $C_i \xleftarrow{R} \mathcal{E}_K(M_b)$
$A_{\text{sfctxt}} \Leftarrow C_i$
Reply to $\mathcal{D}^*_K(C)$ queries as follows:
$j \leftarrow j+1$; $M \leftarrow \mathcal{D}_K(C)$
If $j > i$ or $C \neq C_j$ then phase $\leftarrow 1$
If $M \neq \perp$ and phase $= 1$ then return 1
If $M \neq \perp$ then $A_{\text{sfctxt}} \Leftarrow 1$
Else $A_{\text{sfctxt}} \Leftarrow 0$
Until A_{sfctxt} halts
Return 0

Channel is initially "in sync"

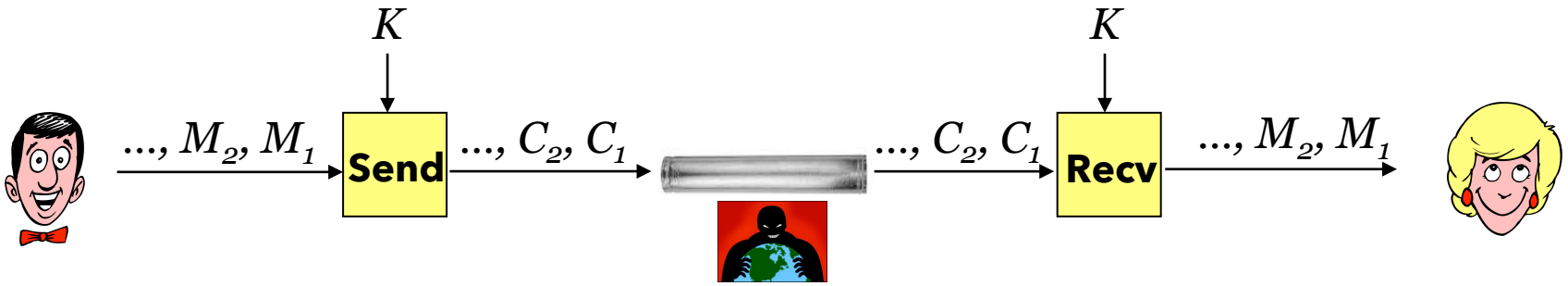
Sequencing of **sent** ciphertexts

Sequencing of **received** ciphertexts

Channel is "out of sync", relative to what has been sent. GAME ON!

Adversary delivered out-of-sync ctxts and decryption failed to catch it! WIN!

"Atomic message" semantics



A secure channel should provide:

Privacy for messages
Integrity protection for the transmission
Protection against out-of-order delivery and replay

So far, all of our security notions assume "atomic messages":

Sender submits one message M_i and this maps to one transmitted ciphertext C_i

Implicitly, the sender is guaranteed that the basic "protected unit" is a message.

Observations

1. The application may pass in arbitrary length data, but an RFC-compliant implementation of **the TLS secure channel may fragment however it likes**, so long as fragments are at most 2^14 bytes long.

2. TLS records embellished with a sequence number (SQN), to protect against out-of-order delivery (which includes replay, fragment dropping)

If encryption is CBC mode, the SQN is not sent.
If encryption is AEAD the SQN may or may not be sent.

Fragmentation and streams

Real protocols and network behaviors cause fragmentation:

One message M_i mapping to an unknown number of transmitted ciphertexts $C_{i,1}$ $C_{i,2}$ … [Boldyreva, Degabriele, Paterson, Stam EC'12]

Fragmentation and streams

Real protocols (e.g. TLS) and network behaviors cause fragmentation:

One message M_i mapping to an unknown number of transmitted ciphertexts $C_{i,1}$ $C_{i,2}$ … [Boldyreva, Degabriele, Paterson, Stam EC'12]

What's more, TLS v1.2 says…

"Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType MAY be coalesced into a single TLSPlaintext record…)"

Thus from the application's perspective, security guarantees are assumed for the entire **stream** of data provided to the TLS record layer.

Stream semantics, not message semantics.

Fragmentation and streams

Real protocols (e.g. TLS) and network behaviors cause fragmentation:

One message M_i mapping to an unknown number of transmitted ciphertexts $C_{i,1}$ $C_{i,2}$ … [Boldyreva, Degabriele, Paterson, Stam EC'12]

What's more, TLS v1.2 says…

"Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType MAY be coalesced into a single TLSPlaintext record…)"

Thus from the application's perspective, security guarantees are assumed for the entire **stream** of data provided to the TLS record layer.

Stream semantics, not message semantics.

..this is reflected in secure channel implementations, e.g. OpenSSL's API:

```
EVP_EncryptInit(&ctx, EVP_aes_256_cbc(), key, iv)
EVP_EncryptUpdate(&ctx, out, &outlen1, in, sizeof(in))
EVP_EncryptFinal(&ctx, out + outlen1, &outlen2)
```

[FGMP] Stream-based Channels

Definition 3.1 (Syntax of stream-based channels). *A* stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated sending and receiving state space* $\mathcal{S}_S$ *resp.* $\mathcal{S}_R$ *and error space* $\mathcal{E}$*, where* $\mathcal{E} \cap \{0,1\}^* = \emptyset$*, consists of three efficient algorithms:*

- Init. *On input a security parameter* 1^λ*, this probabilistic algorithm outputs initial states* $\mathsf{st}_{S,0} \in \mathcal{S}_S$*,* $\mathsf{st}_{R,0} \in \mathcal{S}_R$ *for the sender and the receiver, respectively. We write* $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \leftarrow_\$ \mathsf{Init}(1^\lambda)$.

Init: models initialization of the channel, producing sender- and receiver-side state
(think: keys, counters, buffers, etc.)

[FGMP] Stream-based Channels

Definition 3.1 (Syntax of stream-based channels). *A* stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated sending and receiving state space* $\mathcal{S}_S$ *resp.* $\mathcal{S}_R$ *and error space* $\mathcal{E}$*, where* $\mathcal{E} \cap \{0,1\}^* = \emptyset$*, consists of three efficient algorithms:*

- Send. *On input a state* $\mathsf{st}_S \in \mathcal{S}_S$*, a message fragment* $m \in \{0,1\}^*$*, and a flush flag* $f \in \{0,1\}$*, this (possibly) probabilistic algorithm outputs an updated state* $\mathsf{st}'_S \in \mathcal{S}_S$ *and a ciphertext fragment* $c \in \{0,1\}^*$*. We write* $(\mathsf{st}'_S, c) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_S, m, f)$.

Init: models initialization of the channel, producing sender- and receiver-side state (think: keys, counters, buffers, etc.)

Send: models stateful/randomized transformation of message fragments into (possibly empty) ciphertext fragments. Updates state (e.g. counters, buffers) and observes a "flush" signal.

[FGMP] Stream-based Channels

Definition 3.1 (Syntax of stream-based channels). *A* stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated sending and receiving state space* $\mathcal{S}_S$ *resp.* $\mathcal{S}_R$ *and error space* $\mathcal{E}$*, where* $\mathcal{E} \cap \{0,1\}^* = \emptyset$*, consists of three efficient algorithms:*

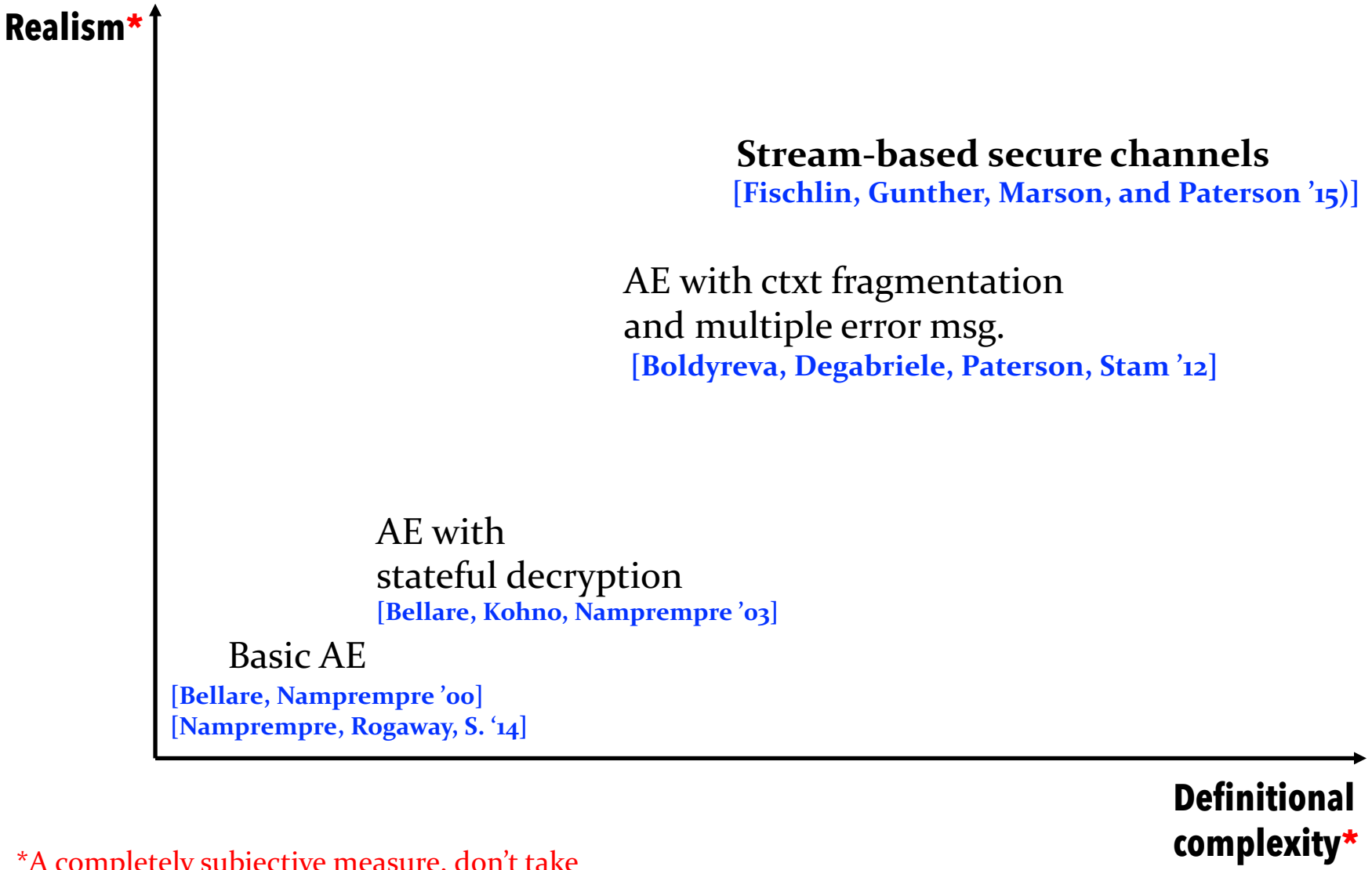
- Recv. *On input a state* $\mathsf{st}_R \in \mathcal{S}_R$ *and a ciphertext fragment* $c \in \{0,1\}^*$*, this deterministic algorithm outputs an updated state* $\mathsf{st}'_R \in \mathcal{S}_R$ *and a message fragment* $m \in (\{0,1\} \cup \mathcal{E})^*$*. We write* $(\mathsf{st}'_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$.

Init: models initialization of the channel, producing sender- and receiver-side state (think: keys, counters, buffers, etc.)

Send: models stateful/randomized transformation of message fragments into (possibly empty) ciphertext fragments. Updates state (e.g. counters, buffers) and observes a "flush" signal.

Recv: models stateful transformation of ciphertext fragments into (possibly empty) message fragments or error symbols. Updates state (e.g. counters, buffers).

"Data is a stream" [FGMP]



*A completely subjective measure, don't take this too literally...

Integrity of streams [FGMP (full version)]

$\mathsf{Expt}^{\text{INT-atk}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:

1 $(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
2 $\mathsf{sync} \leftarrow 1$
3 $\mathsf{win} \leftarrow 0$
4 $M_S, C_S \leftarrow \varepsilon,\ M_R, C_R \leftarrow \varepsilon$
5 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
6 return win

Integrity of streams [FGMP (full version)]

$\mathsf{Expt}^{\text{INT-atk}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:

1 $(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
2 $\mathsf{sync} \leftarrow 1$
3 $\mathsf{win} \leftarrow 0$
4 $M_S, C_S \leftarrow \varepsilon,\ M_R, C_R \leftarrow \varepsilon$
5 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
6 return win

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m, f)$:

7 $(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_S, m, f)$
8 $M_S \leftarrow M_S \,\|\, m$
9 $C_S \leftarrow C_S \,\|\, c$
10 return c to $\mathcal{A}$

append message fragment to sender-side message stream

append ciphertext fragment to sender-side ciphertext stream

Integrity of streams [FGMP (full version)]

$\mathsf{Expt}^{\mathsf{INT\text{-}atk}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:

1 $(\mathsf{st}_S, \mathsf{st}_R) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$
2 $\mathsf{sync} \leftarrow 1$
3 $\mathsf{win} \leftarrow 0$
4 $M_S, C_S \leftarrow \varepsilon,\ M_R, C_R \leftarrow \varepsilon$
5 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
6 return win

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m, f)$:

7 $(\mathsf{st}_S, c) \leftarrow\!\!\$\ \mathsf{Send}(\mathsf{st}_S, m, f)$
8 $M_S \leftarrow M_S \parallel m$
9 $C_S \leftarrow C_S \parallel c$
10 return c to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:

16 if $\mathsf{sync} = 0$ then // already out-of-sync
17 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
18 if $m \notin \mathcal{E}^*$ then $\mathsf{win} \leftarrow 1$
19 else if $C_R \parallel c \preccurlyeq C_S$ then // still in-sync
20 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
21 $C_R \leftarrow C_R \parallel c$
22 else
23 if $C_R \prec [C_R \parallel c, C_S]$ then
// c deviates or exceeds, contains genuine part
24 $\widetilde{c} \leftarrow [C_R \parallel c, C_S] \,\%\, C_R$
25 $\widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
26 $(\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
27 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
28 $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
29 else // c deviates or exceeds, contains no genuine part
30 $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
31 $m \leftarrow m'$
32 if $C_S \not\preccurlyeq C_R \parallel c$ or $m' \neq \varepsilon$ then
// deviation, or exceeding portion produces output
33 $\mathsf{sync} \leftarrow 0$
34 $C_R \leftarrow C_R \parallel c$
35 if $m' \notin \mathcal{E}^*$ then $\mathsf{win} \leftarrow 1$
36 return m to $\mathcal{A}$

Why so complicated?

Intuition for stateful “atomic” AE [BKN]:

“If the receiver detects an out-of-sync ciphertext, throw an error!”

Why so complicated?

Intuition for stateful “atomic” AE [BKN]:

“If the receiver detects an out-of-sync ciphertext, throw an error!”

Intuition for stream-based setting [FGMP]:

“**WHEN** the receiver detects an out-of-sync ciphertext, throw an error!”

Why the distinction?

Why so complicated?

Intuition for stateful "atomic" AE [BKN]:

"If the receiver detects an out-of-sync ciphertext, throw an error!"

Intuition for stream-based setting [FGMP]:

"**WHEN** the receiver detects an out-of-sync ciphertext, throw an error!"

Why the distinction?

1. **Fragmentation**! Even under honest operation, protocol and network may fragment

Why so complicated?

Intuition for stateful "atomic" AE [BKN]:

"If the receiver detects an out-of-sync ciphertext, throw an error!"

Intuition for stream-based setting [FGMP]:

"**WHEN** the receiver detects an out-of-sync ciphertext, throw an error!"

Why the distinction?

1. **Fragmentation**! Even under honest operation, protocol and network may fragment
2. **Fragmentation**! Adversary controls the network, and can refragment, reorder, etc.

Receiver may have to accept many **adversarially delivered ciphertext fragments** before it can actually determine that the honest ciphertext stream is out-of-sync

The full version of [FGMP] is a lovely paper.

Go read it.

What is a “partially specified secure channel”?!

Real specification documents, again…

Real secure channel specifications are full of MUSTs, MUST NOTs, SHOULDs, SHOULD NOTs, and MAYs

(61 of them in TLS v 1.2)

MUST = absolute requirement
SHOULD = recommended, but there are valid reasons you might decide not to do this
MAY = optional

> Whenever an implementation encounters a condition which is defined as a fatal alert, it MUST send the appropriate alert prior to closing the connection. For all errors where an alert level is not explicitly specified, the sending party MAY determine at its discretion whether to treat this as a fatal error or not.

Real specification documents, again…

Real secure channel specifications are full of MUSTs, MUST NOTs, SHOULDs, SHOULD NOTs, and MAYs (61 of them in TLS v 1.2)

MUST = absolute requirement
SHOULD = recommended, but there are valid reasons you might decide not to do this
MAY = optional

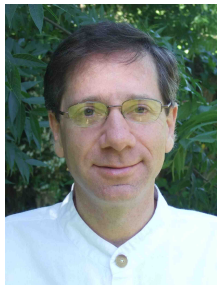
> Whenever an implementation encounters a condition which is defined as a fatal alert, it MUST send the appropriate alert prior to closing the connection. For all errors where an alert level is not explicitly specified, the sending party MAY determine at its discretion whether to treat this as a fatal error or not.

Real secure channel specifications are vague or silent on many important implementation details

> "Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType MAY be coalesced into a single TLSPlaintext record…)"

↑ No mention of how

Words of caution, in search of a formal solution



"Because of all this complexity, we would claim that nobody has ever proven the cryptographic security of something like the "full" TLS 1.2; instead, one extracts some piece and goes from there"

"Abstraction is what we do. But, at another level, [...] if you prove something about the (self-identified) cryptographic core of an authentication protocol, does this actually prove anything about the full-fledged scheme? An approach in which security-relevant features of real-world protocols are routinely elided in corresponding analyses ought to raise foundational concerns."

Authentication without Elision:
Partially Specified Protocols, Associated Data, and Cryptographic Models Described by Code
(Rogaway and Stegers, CSF'09)

"Specifications" = families of implementations

The Transport Layer Security (TLS) Protocol
Version 1.2

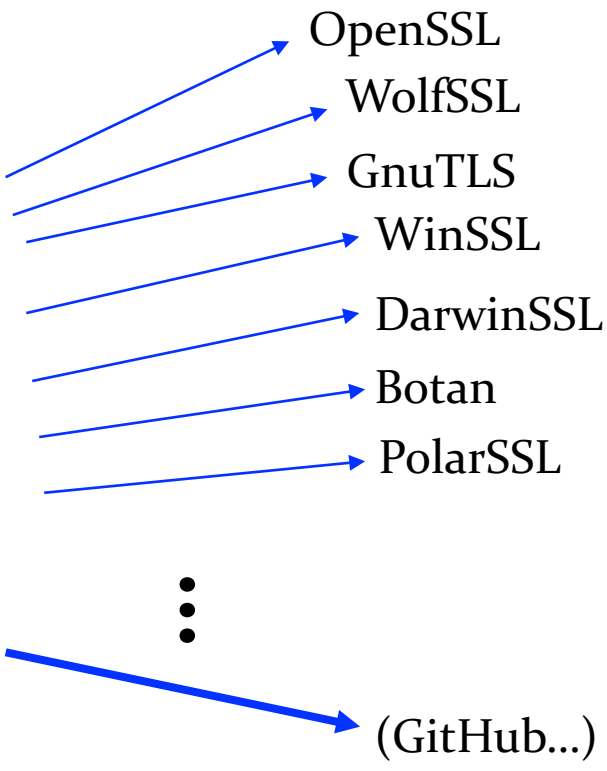
Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document specifies Version 1.2 of the Transport Layer Security (TLS) protocol. The TLS protocol provides communications security over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Table of Contents



Basic idea of the "partially specified X" viewpoint

Real world specification = **Partial specification of things that absolutely must be defined, implemented, verified correct**

\+

a collection of optional behaviors, unspecified details

Partially Specified Channels

[FGMP] left open the extention of stream-based channels to the **multiplexed-stream setting,** something explicit in TLS v1.3 and useful in practice.

A **partially specified channel** (PSC) with support for multiplexing is a 5-tuple

Init^{O} intializes state for **Mux, Write, Read, Demux**

Mux^{O} the stateful multiplexing algorithm takes as input a *plaintext fragment M*, *stream context sc*, and returns a *channel fragment* X, its *context* H, and some *auxiliary output* α

Write^{O} the stateful channel-writing algorithm takes a *channel fragment X* , *context H* , and *auxiliary information* α, and produces a *ciphertext fragment C* and a *status message* γ

Send

Read^{O} The stateful channel-reading algorithm takes a *ciphertext fragment C*, and returns a *ciphertext fragment* Y, its *context* H, and *auxiliary output* α.

Demux^{O} The stateful demultiplexing algorithm takes a *ciphertext fragment Y* with *channel context H*, *auxiliary information* α, and returns a *plaintext fragment M* with *stream context sc*, along with a status message γ .

Recv

Partially Specified Channels

What's with the oracle? It handles all of the things **that are not explicitly specified by the implementation** of the named algorithm.

"Specification details" oracle

Init^{O} intializes state for **Mux, Write, Read, Demux**

Mux^{O} the stateful multiplexing algorithm takes as input a *plaintext fragment M*, *stream context sc*, and returns a *channel fragment* X, its *context* H, and some *auxiliary output* α

Write^{O} the stateful channel-writing algorithm takes a *channel fragment X* , *context H* , and *auxiliary information* α, and produces a *ciphertext fragment C* and a *status message* γ

Send

Read^{O} The stateful channel-reading algorithm takes a *ciphertext fragment C*, and returns a *ciphertext fragment* Y, its *context* H, and *auxiliary output* α.

Demux^{O} The stateful demultiplexing algorithm takes a *ciphertext fragment Y* with *channel context H*, *auxiliary information* α, and returns a *plaintext fragment M* with *stream context sc*, along with a status message γ .

Recv

Fully Specified Channels

A PSC becomes a "fully specified channel" once you fix the behavior of the specification details oracle.

Init^{O} intializes state for **Mux, Write, Read, Demux**

Mux^{O} the stateful multiplexing algorithm takes as input a *plaintext fragment M*, *stream context sc*, and returns a *channel fragment* X, its *context* H, and some *auxiliary output* α

Write^{O} the stateful channel-writing algorithm takes a *channel fragment X* , *context H* , and *auxiliary information* α, and produces a *ciphertext fragment C* and a *status message* γ

Send

Read^{O} The stateful channel-reading algorithm takes a *ciphertext fragment C*, and returns a *ciphertext fragment* Y, its *context* H, and *auxiliary output* α.

Demux^{O} The stateful demultiplexing algorithm takes a *ciphertext fragment Y* with *channel context H*, *auxiliary information* α, and returns a *plaintext fragment M* with *stream context sc*, along with a status message γ .

Recv

Adversarial Fully Specified Channels

Here's the cool part:

In the security notions, the adversary plays it's game as usual,
but whenever a SD-oracle call is made, the adversary services it.

$\mathbf{Exp}^{\text{int-cs}}_{\mathcal{CH}}(\mathcal{A})$

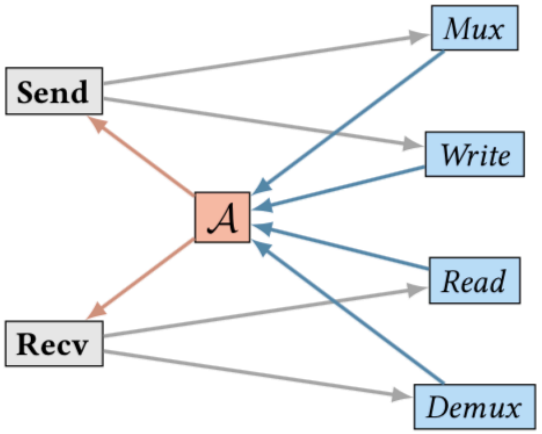
1 **declare str** S, **bool** $sync$, win
2 $(Mu, Wr, Re, De) \twoheadleftarrow Init()$
3 $sync \leftarrow 1$; $\mathcal{A}^{\mathbf{Send},\mathbf{Recv}}$
4 **return** win

Send(M, sc)

5 $(X, H, \alpha) \twoheadleftarrow Mux^{\mathcal{A}}(M, sc, \mathbf{var}\ Mu)$
6 $(C, \gamma) \twoheadleftarrow Write^{\mathcal{A}}(X, H, \alpha, \mathbf{var}\ Wr)$
7 $S \leftarrow S \,\|\, C$
8 **return** (C, γ)

Recv(C)

9 $(Y, H, \alpha) \twoheadleftarrow Read^{\mathcal{A}}(C, \mathbf{var}\ Re)$
10 $(M, sc, \gamma) \twoheadleftarrow Demux^{\mathcal{A}}(Y, H, \alpha, \mathbf{var}\ De)$
11 **if** $sync$ **and** $Y \preceq S$ **then** $S \leftarrow S \,\%\, Y$
12 **else** $sync \leftarrow 0$
13 $win \leftarrow win \vee (M \neq \bot \wedge sc \neq \bot)$
14 **return** (M, sc, γ)



Security proof means, loosely,
"secure in the presence of worst-case implementation of specification details"

Interesting things about PSCs

We use the PSC viewpoint to capture and analyze the TLS v 1.3 record layer as-is, and not some imagined realization of it.

For a PSC: integrity of ciphertext streams *does not imply* integrity of plaintext streams.

You can't define correctness in a satisfying way until you **fully** specify.

(Correctness for a real protocol might be very hard to prove!)

For a FSC: integrity of ciphertext streams *does* integrity of plaintext streams.

Many other things! **Seems like a very powerful and useful viewpoint**.

(Partially Specified) Secure Channels

Tom Shrimpton
University of Florida