Authenticated Encryption and Secure Channels – There and Back Again

Kenny Paterson

Information Security Group

@kennyog; www.isg.rhul.ac.uk/~kp

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Overview

- Secure channels and their properties
- AEAD – your local cryptographer's abstraction
- AEAD ≠ secure channel
- From AEAD to secure channels
- Are we there yet?

Secure channels and their properties

Why do we need secure channels?

- Secure communications is the most common real-world application of cryptography today.
    - No, it's not MPC for sugar beet auctions!
- Secure channels are extremely widely-deployed in practice:
    - SSL/TLS, DTLS, IPsec, SSH, OpenVPN,…
    - WEP/WPA/WPA2
    - GSM/UMTS/LTE
    - Cryptocat, OTR, SilentCircle
    - OpenPGP, Telegram, Signal, and a thousand other messaging apps
    - QUIC, MinimalT, TCPcrypt

Security properties

- **Confidentiality** – privacy for data
- **Integrity** – detection of data modification
- **Authenticity** – assurance concerning the source of data

Some less obvious security properties

- **Anti-replay**
  - Detection that messages have been repeated
- **Drop-detection**
  - Detection that messages have been deleted by the adversary or dropped by the network.
- **Prevention of re-ordering**
  - Preserving the relative order of messages in *each* direction.
  - Preserving the relative order of messages sent and received in *both* directions.
- **Prevention of traffic-analysis.**
  - Using traffic padding and length-hiding techniques.

Possible functionality requirements

- **Speedy**
- **Low-memory**
- **On-line/parallelisable crypto-operations**
    - Performance is heavily hardware-dependent.
    - May have different algorithms for different platforms.
- **IPR-friendly**
    - This issue has slowed down adoption of many otherwise good algorithms, e.g. OCB.
- **Easy to implement**
    - Without introducing any side-channels.

Additional requirements

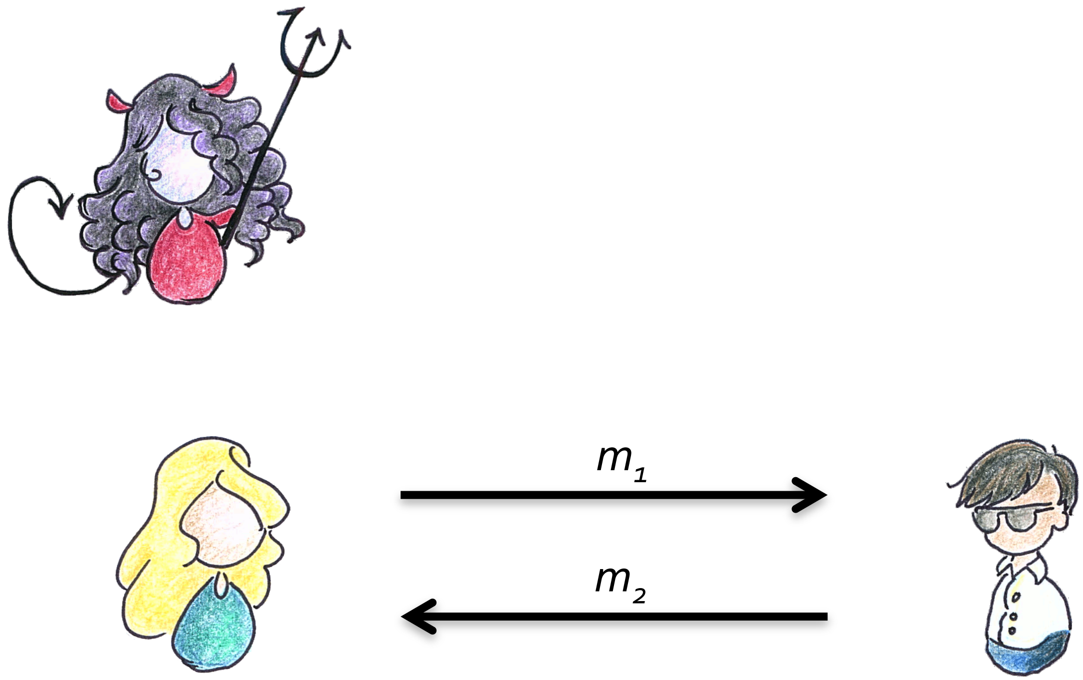
- We need a clean and well-defined API
- Because the reality is that our secure channel protocol will probably be used blindly by a security-naïve developer
- Developers want to "open" and "close" secure channels, and issue "send" and "recv" commands
- They'd like to simply replace TCP with a "secure TCP" having the same API
- Or to just have a blackbox for delivering messages securely

Additional API-driven requirements

- Does the channel provide a stream-based functionality or a message-oriented functionality?
- Does the channel accept messages of arbitrary length and perform its own fragmentation and reassembly, or is there a maximum message length?
- How is error handling performed? Is a single error fatal, leading to tear-down of channel, or is the channel tolerant of errors?
- How are these errors signalled to the calling application? How should the programmer handle them?
- Does the secure channel itself handle retransmissions? Or is this left to the application? Or is it guaranteed by the underlying network transport?
- Does the channel offer data compression?
- These are design choices that all impact on security
- **They are not well-reflected in security definitions for symmetric encryption**

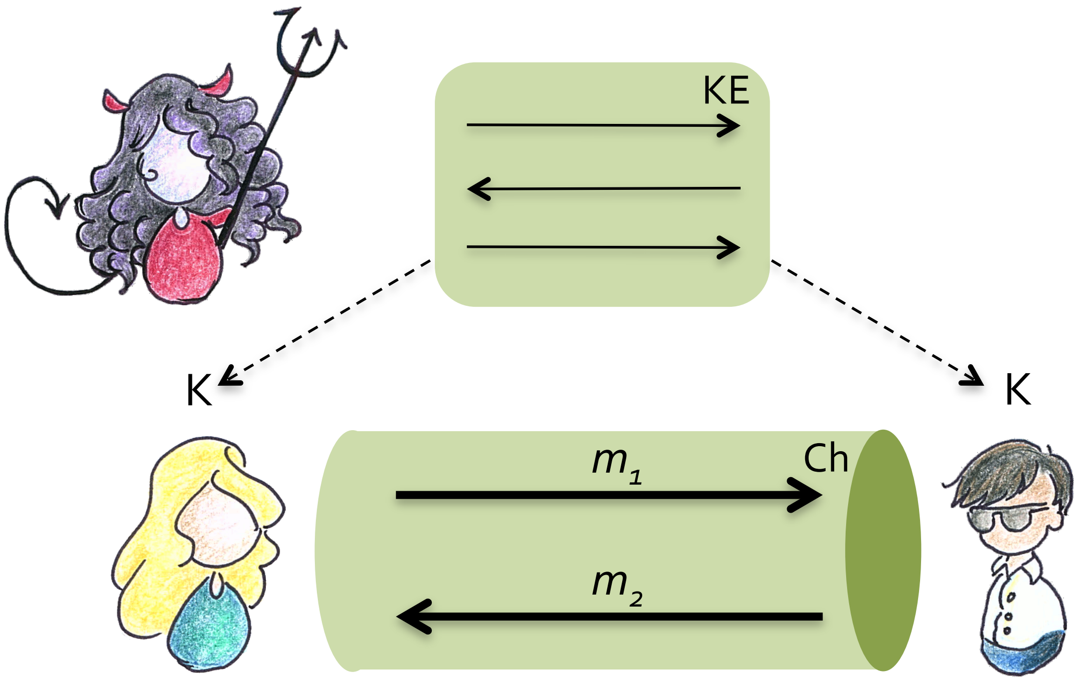
AEAD

Security for Symmetric Encryption



Pictures by Giorgia Azzurra Marson

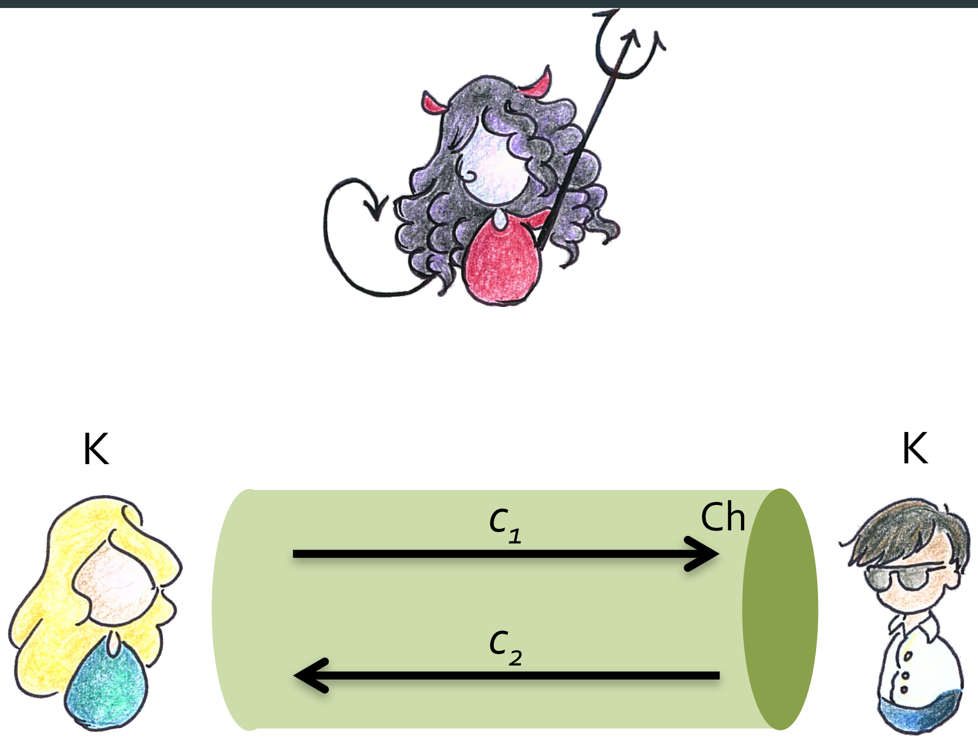
Security for Symmetric Encryption



Security for Symmetric Encryption

$c_1 = \mathrm{Enc}_K(m_1)$

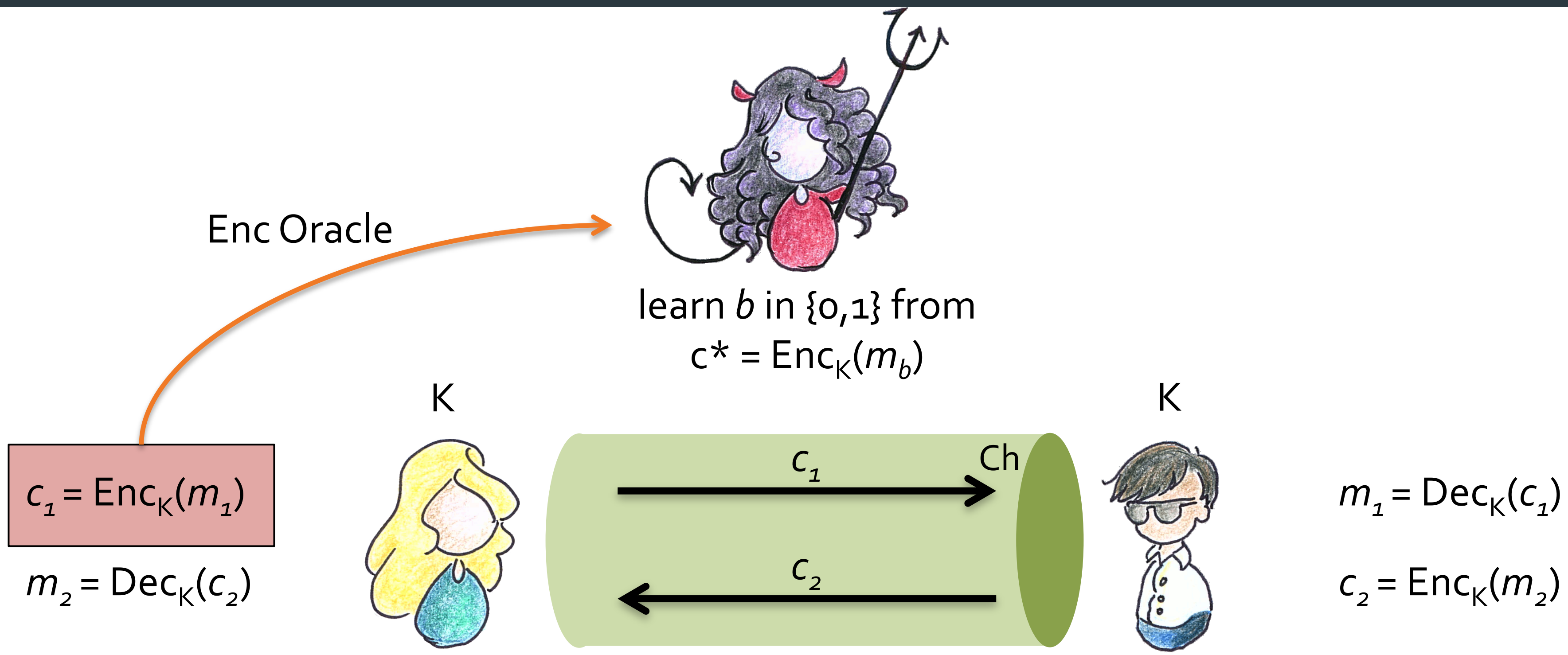
$m_2 = \mathrm{Dec}_K(c_2)$



$m_1 = \mathrm{Dec}_K(c_1)$

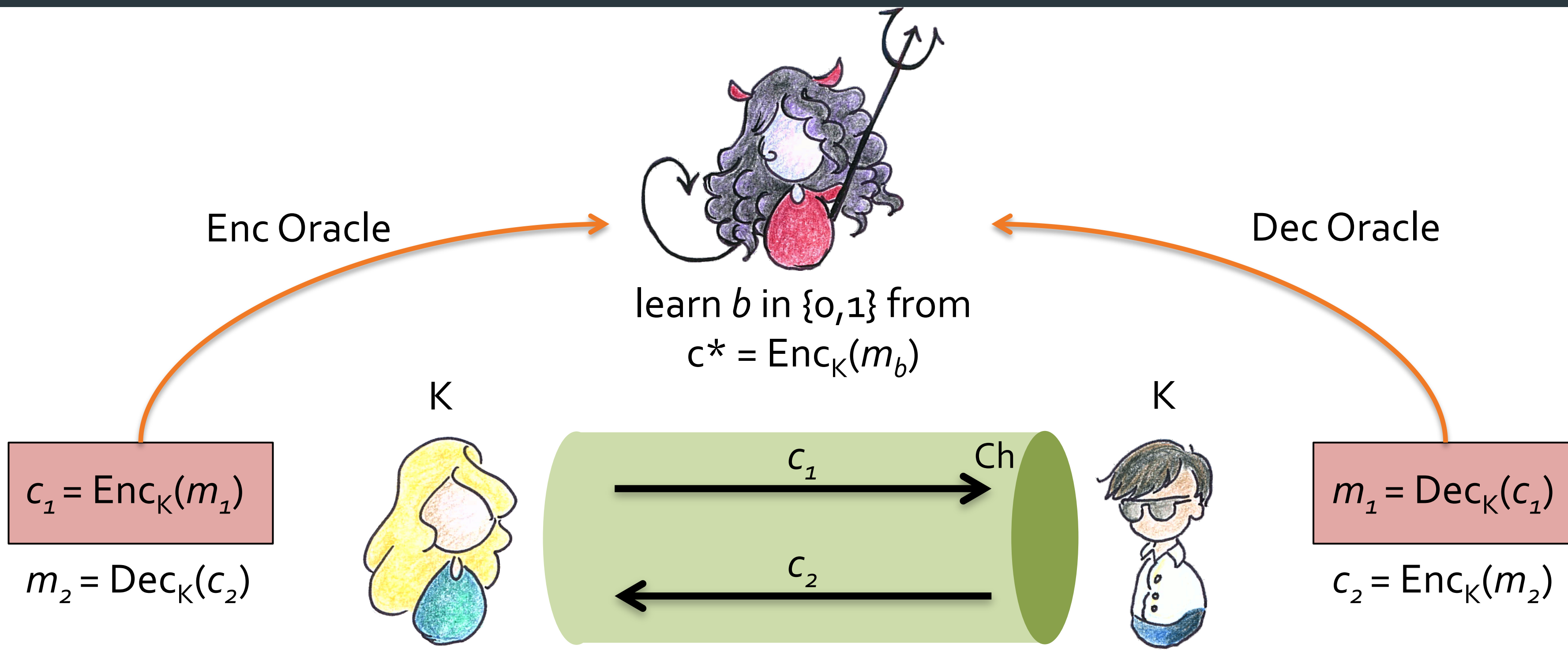
$c_2 = \mathrm{Enc}_K(m_2)$

Security for Symmetric Encryption – Confidentiality



IND-CPA
(Goldwasser-Micali, 1984;
Bellare-Desai-Jokipii-Rogaway, 1997).

Security for Symmetric Encryption – Confidentiality



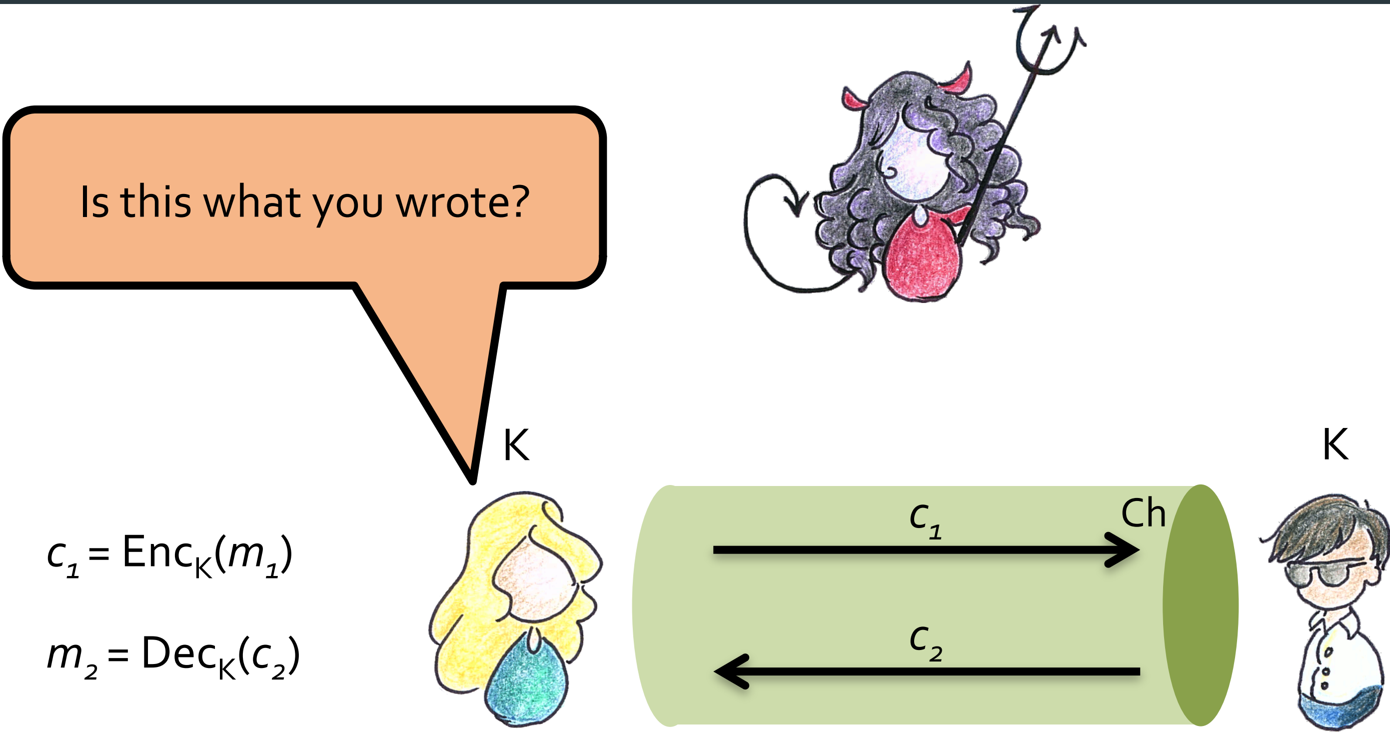
IND-CPA
(Goldwasser-Micali, 1984;
Bellare-Desai-Jokipii-Rogaway, 1997).

IND-CCA
(Naor-Yung, 1990;
Rackoff-Simon, 1997).

Security for Symmetric Encryption – Integrity

$c_1 = \text{Enc}_K(m_1)$

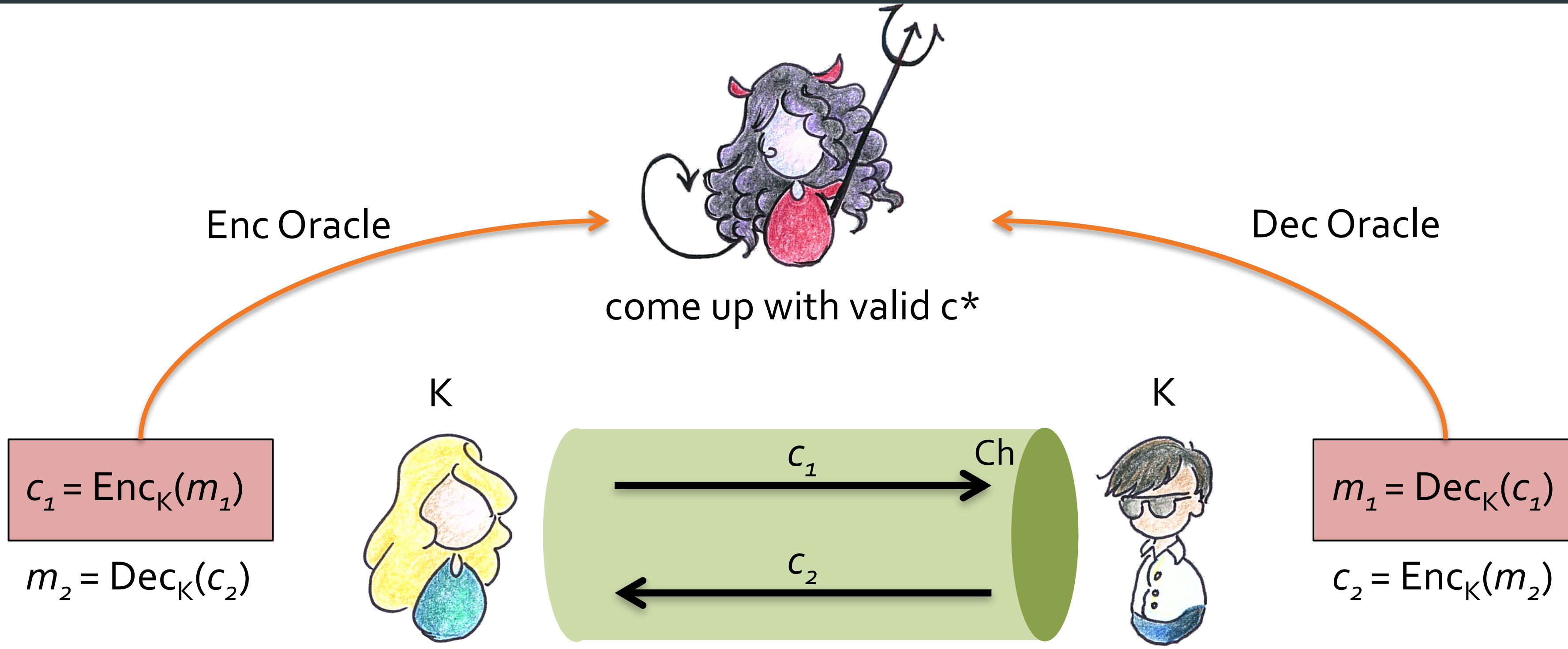
$m_2 = \text{Dec}_K(c_2)$



$m_1 = \text{Dec}_K(c_1)$

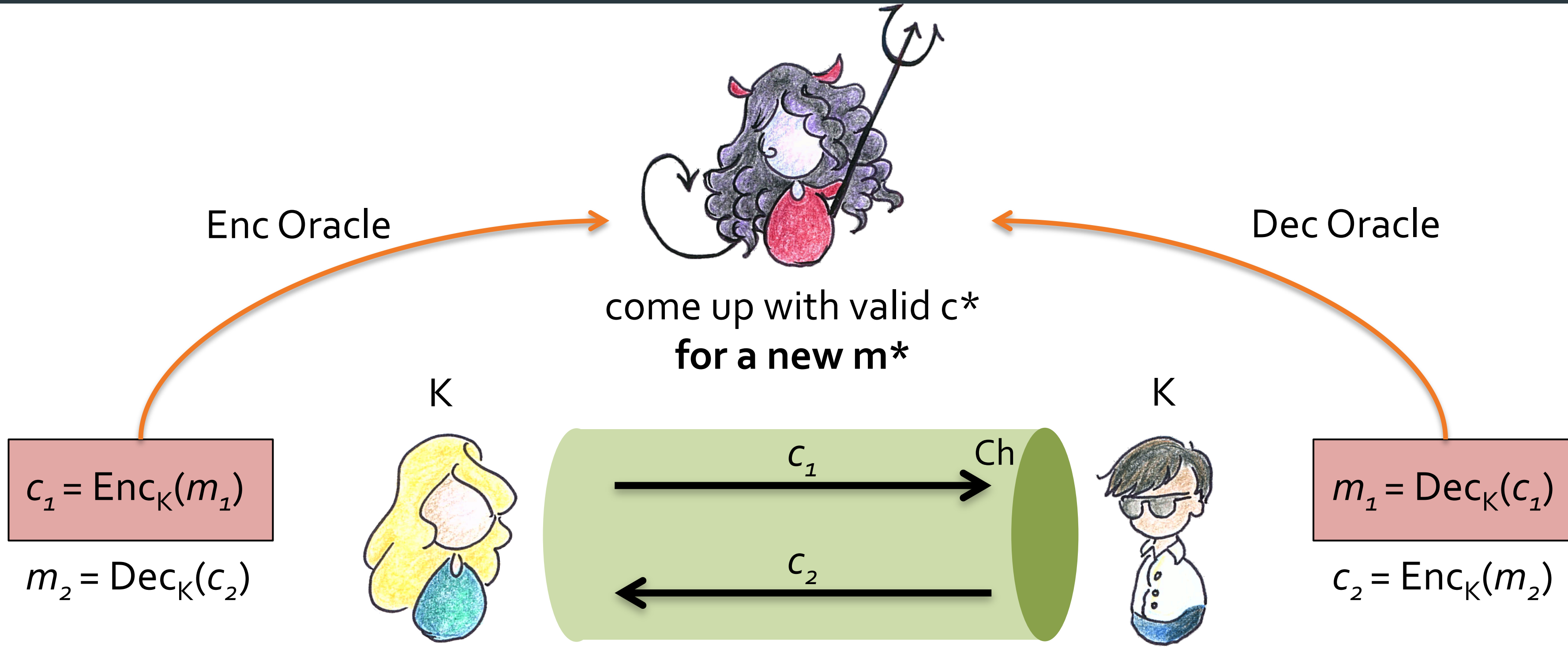
$c_2 = \text{Enc}_K(m_2)$

Security for Symmetric Encryption – Integrity



INT-CTXT
(Bellare, Rogaway, 2000)

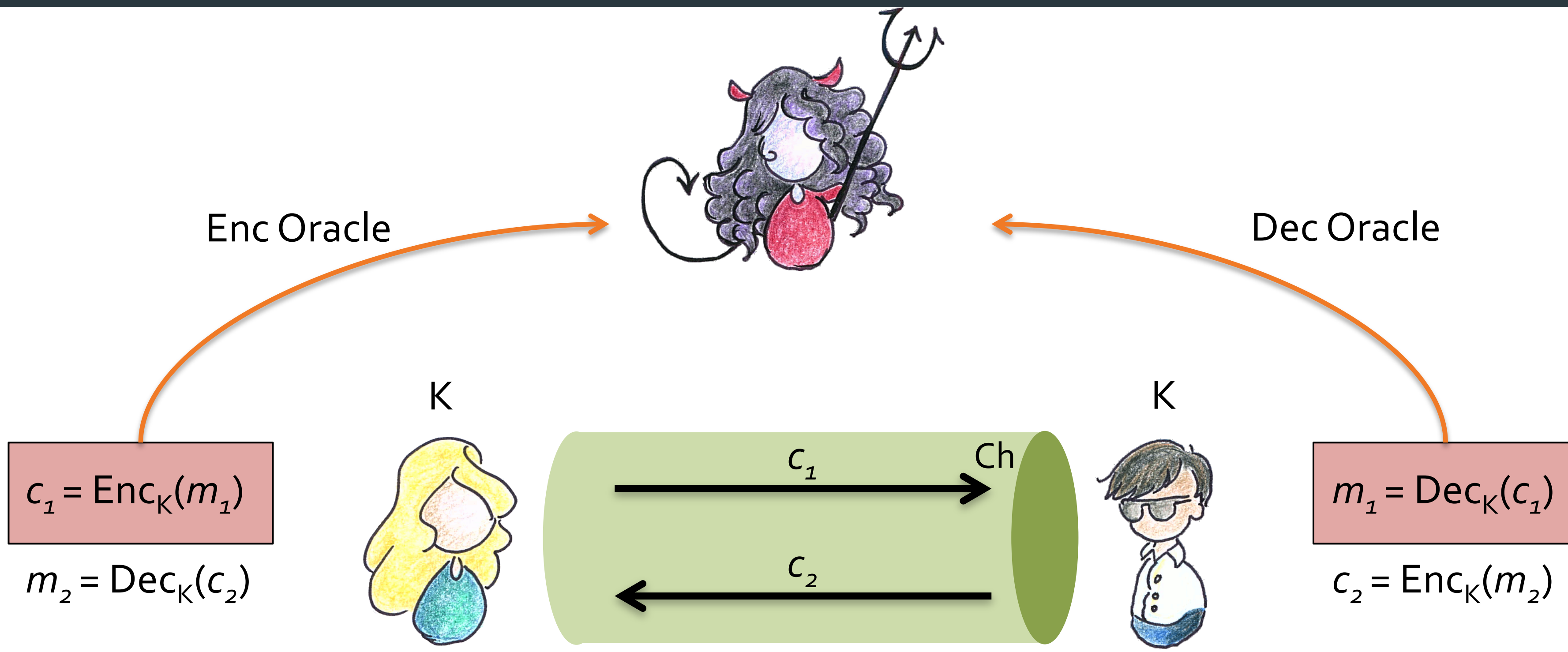
Security for Symmetric Encryption – Integrity



INT-PTXT
(Bellare-Namprempre, 2000)

INT-CTXT
(Bellare, Rogaway, 2000)

Security for Symmetric Encryption – AE

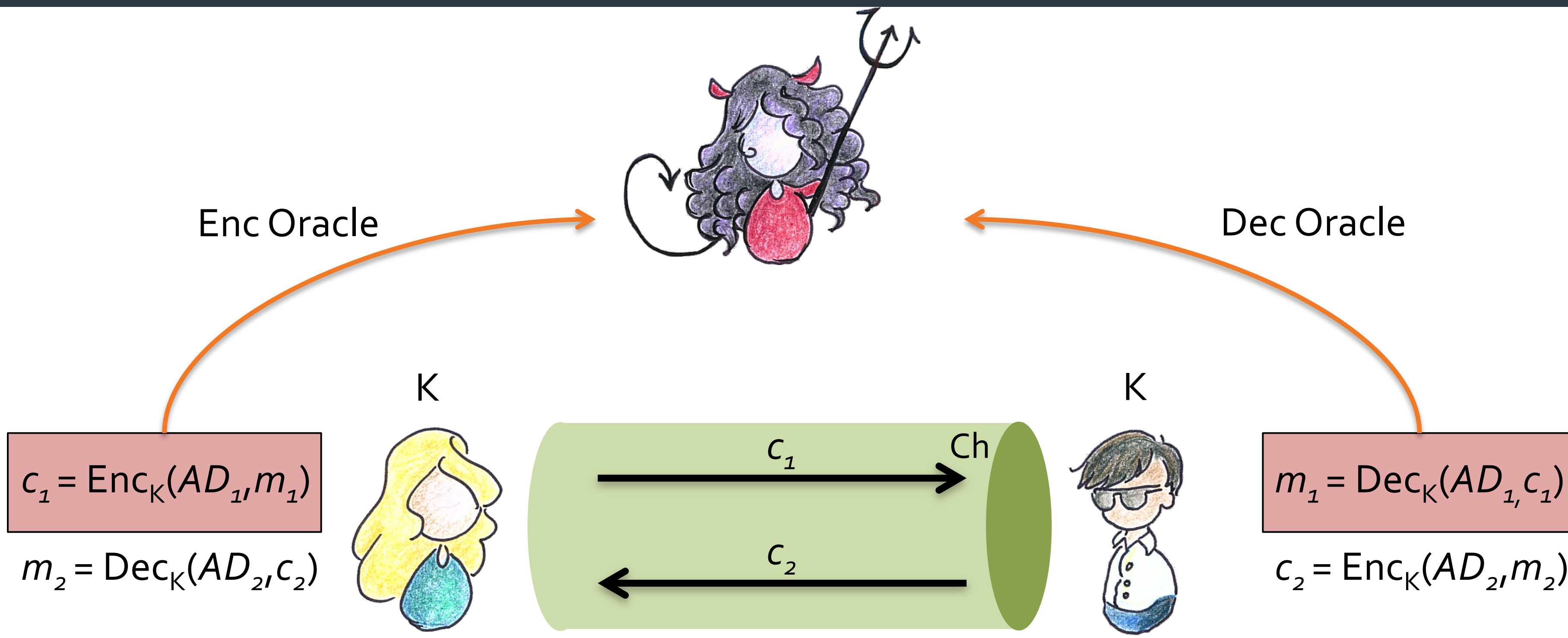


INT-PTXT
(Bellare-Namprempre, 2000)

Authenticated Encryption
IND-CPA + INT-CTXT
(➔IND-CCA)

INT-CTXT
(Bellare, Rogaway, 2000)

Security for Symmetric Encryption – AEAD



Authenticated Encryption with Associated Data

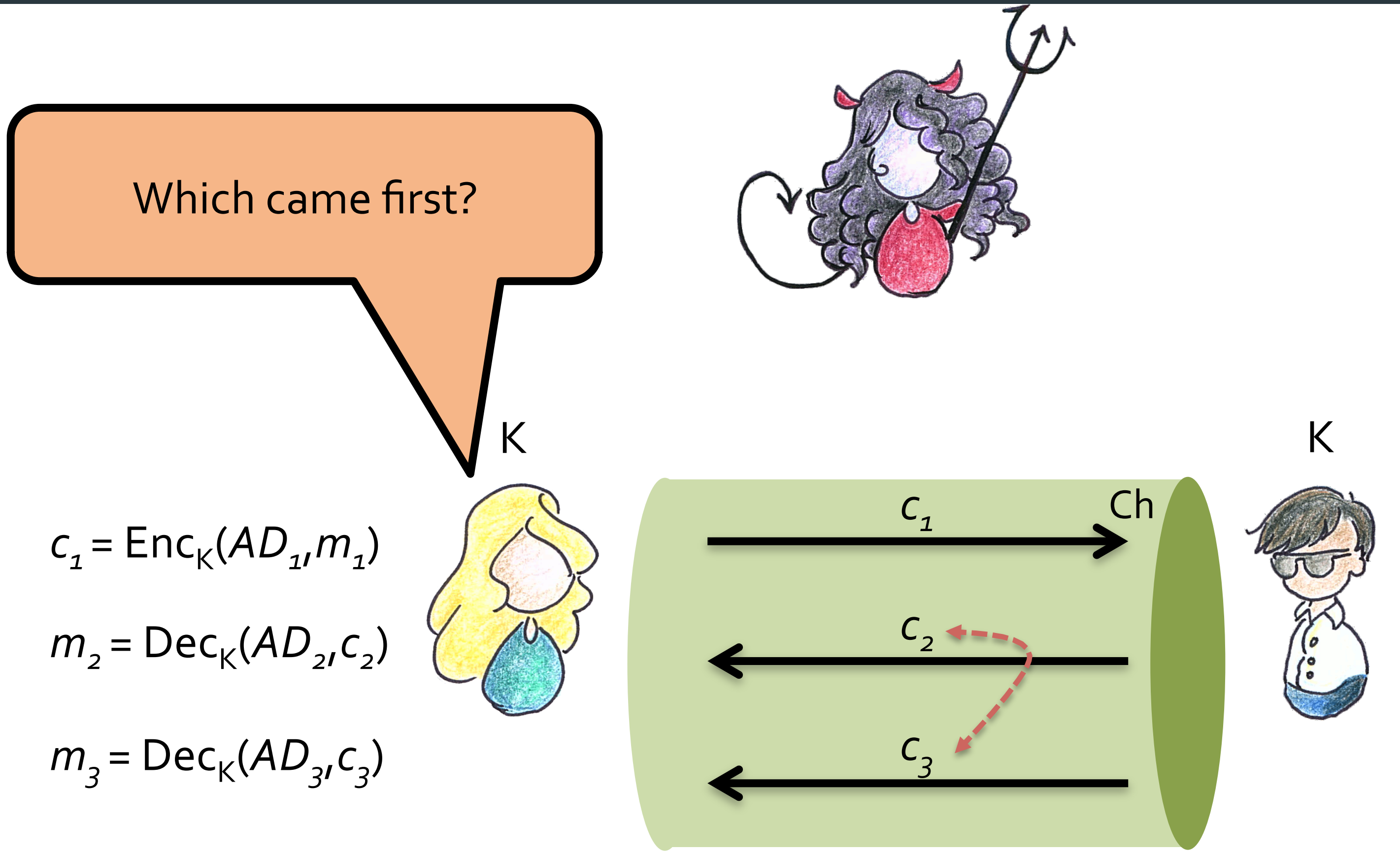
AE security for message *m*

Integrity for associated data *AD*

Strong binding between *c* and *AD*

(Rogaway 2002)

Security for Symmetric Encryption – stateful AEAD



$c_1 = \mathrm{Enc}_K(AD_1, m_1)$

$m_2 = \mathrm{Dec}_K(AD_2, c_2)$

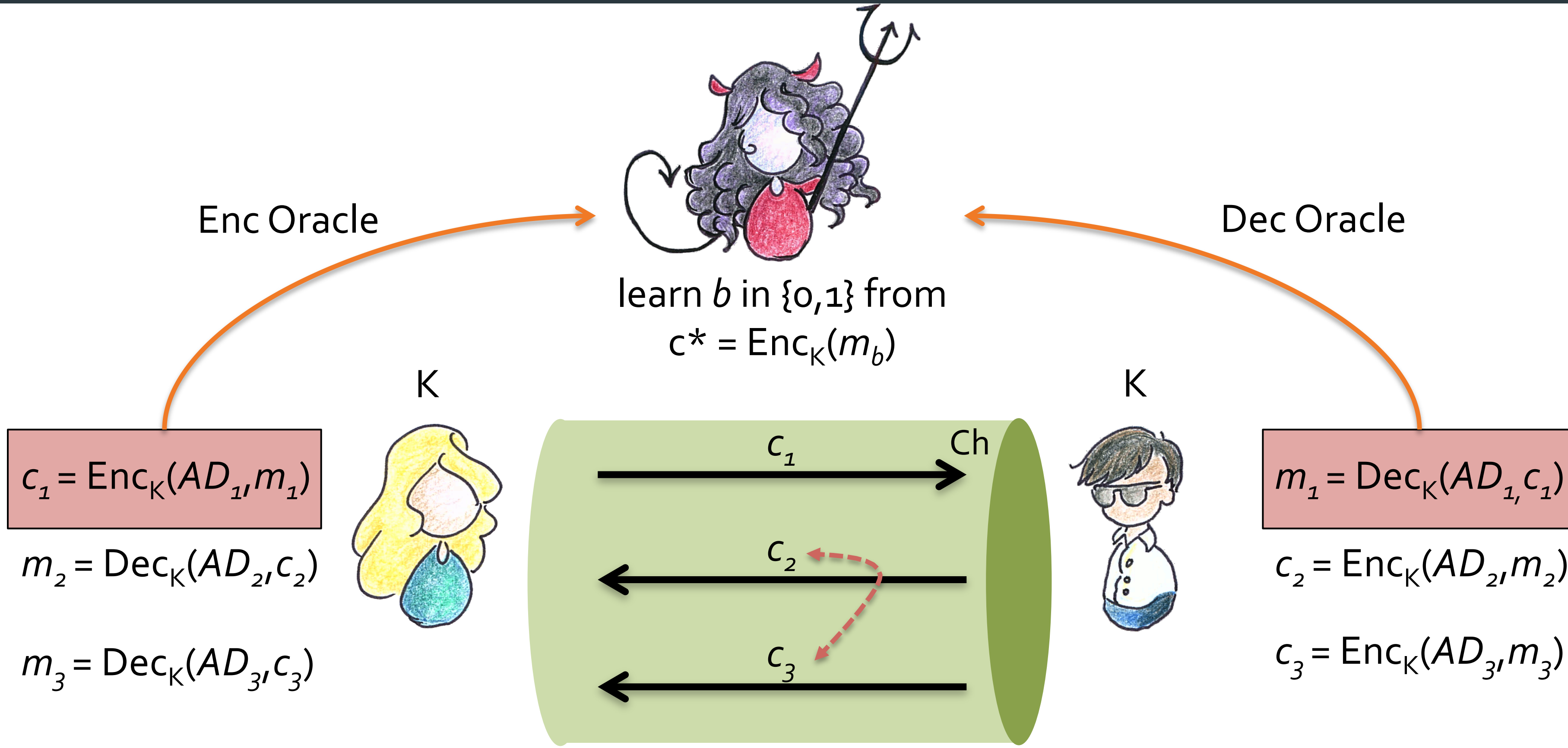
$m_3 = \mathrm{Dec}_K(AD_3, c_3)$

$m_1 = \mathrm{Dec}_K(AD_1, c_1)$

$c_2 = \mathrm{Enc}_K(AD_2, m_2)$

$c_3 = \mathrm{Enc}_K(AD_3, m_3)$

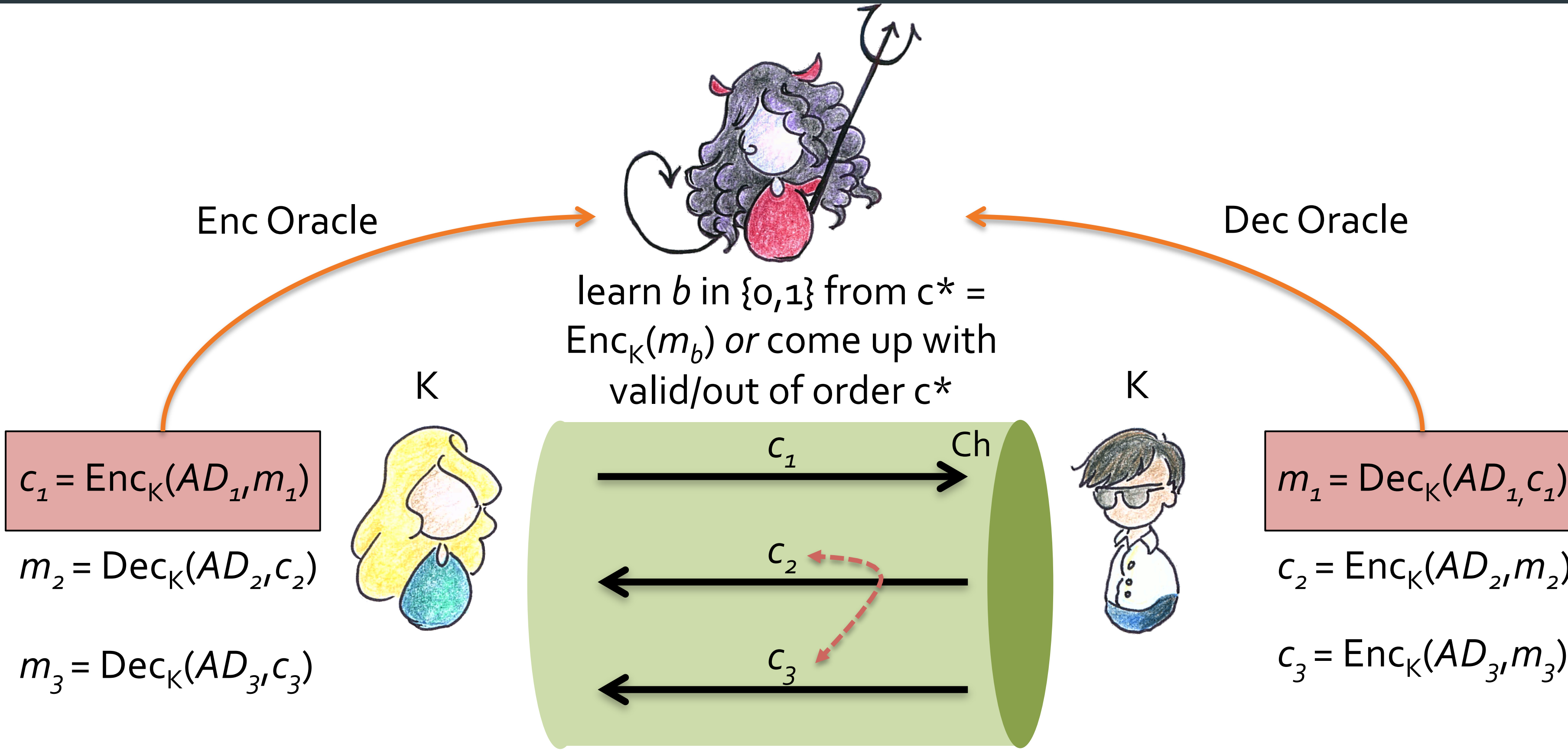
Security for Symmetric Encryption – stateful AEAD



IND-sfCCA

(Bellare-Kohno-Namprempre, 2002)

Security for Symmetric Encryption – stateful AEAD



Stateful AEAD

IND-sfCCA

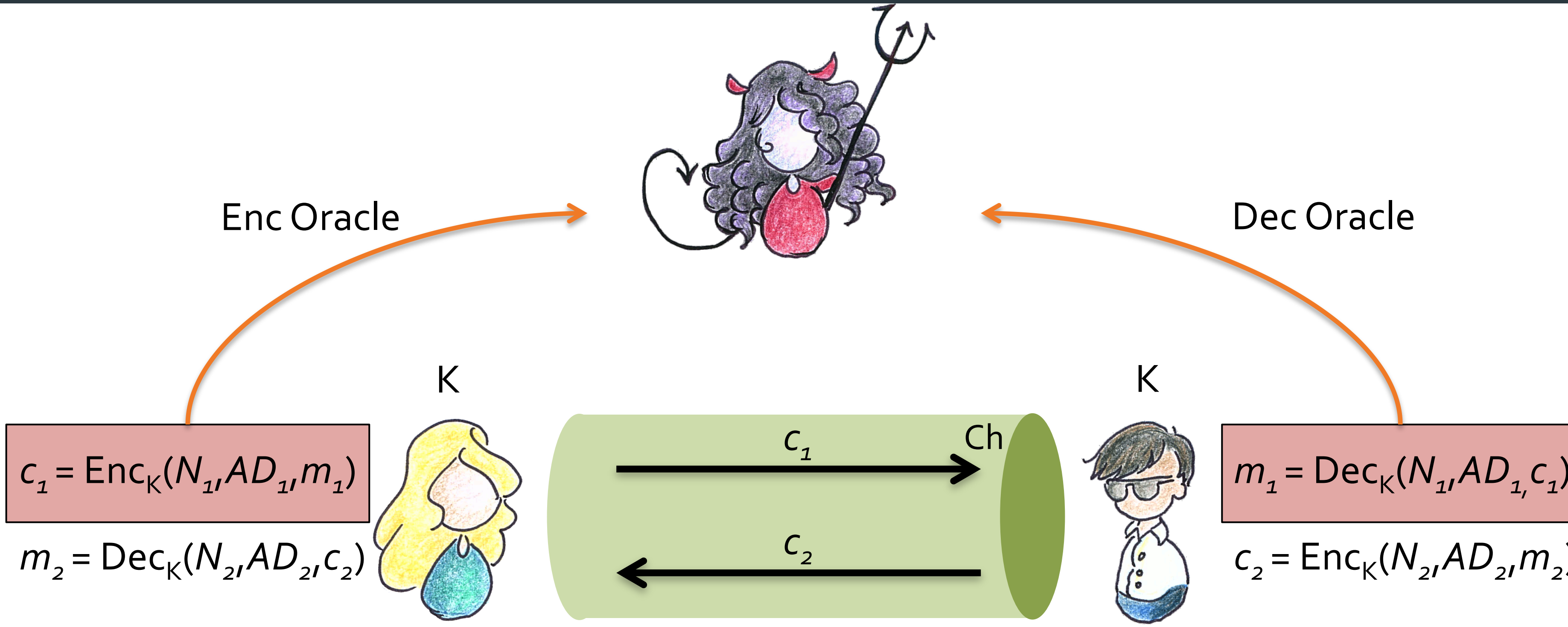
INT-sfCTXT

(Bellare-Kohno-Namprempre, 2002)

INT-sfPTXT

(Brzuska-Smart-Warinschi-Watson, 2013)

Security for Symmetric Encryption – nonce-based AEAD



Nonce-based Authenticated Encryption with Associated Data

As per AEAD, but with additional input *N* to Enc and Dec algorithms

Adversary may arbitrarily specify *N*, but "no repeats" rule

Enc and Dec can now be *stateless* and *deterministic*

(Rogaway 2004)

Security for Symmetric Encryption – further notions

- LH-(stateful)AE(AD)
  - On top of everything else, ciphertexts provide a modicum of hiding of plaintext lengths
  - cf variable length padding in SSL/TLS
  - Introduced by Paterson-Ristenpart-Shrimpton, 2011
  - Incorporated into ACCE framework by Jager-Kohlar-Schage-Schwenk, 2012

CAESAR

- CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness
- Initiated by Dan Bernstein, supported by committee of experts
- Main goal is the design of a *portfolio* of **AE schemes**
- CAEASR has consumed hundreds of person-years of effort and led to a major uptick in research activity

- **It seems that the cryptographic community has settled on nonce-based AE/AEAD as their working abstraction**

AEAD $\neq$ secure channel

AEAD ≠ secure channel

- Recall our application developer:
  - He wants a drop-in replacement for TCP that's secure
  - Actually, he might *just* want to send and receive some atomic messages and not a TCP-like stream
- To what extent does AEAD meet this requirement?
- It doesn't…

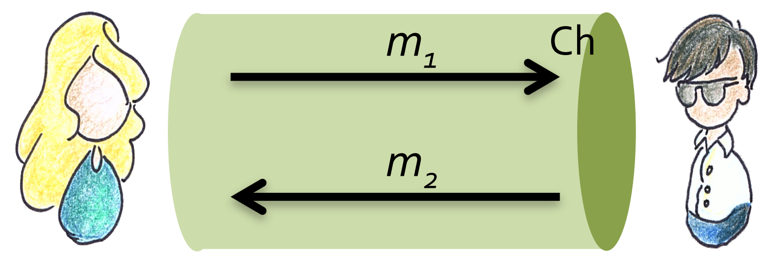
AEAD ≠ secure channel

Enc(.,.,.)

+

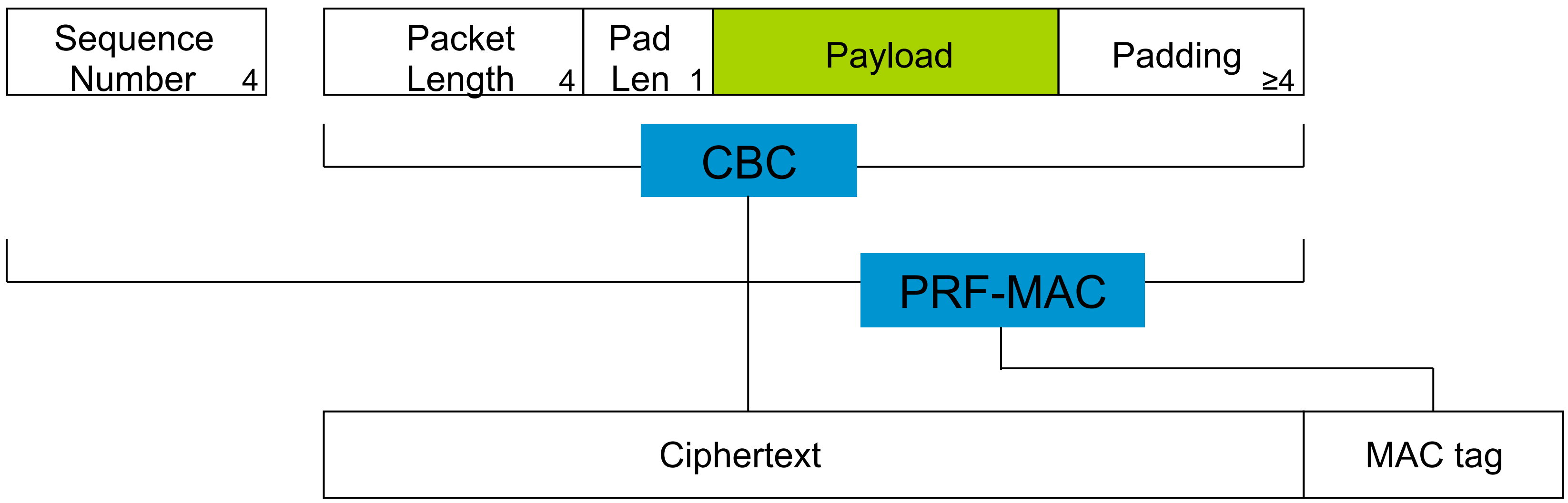
Dec(.,.,.)

MIND THE GAP



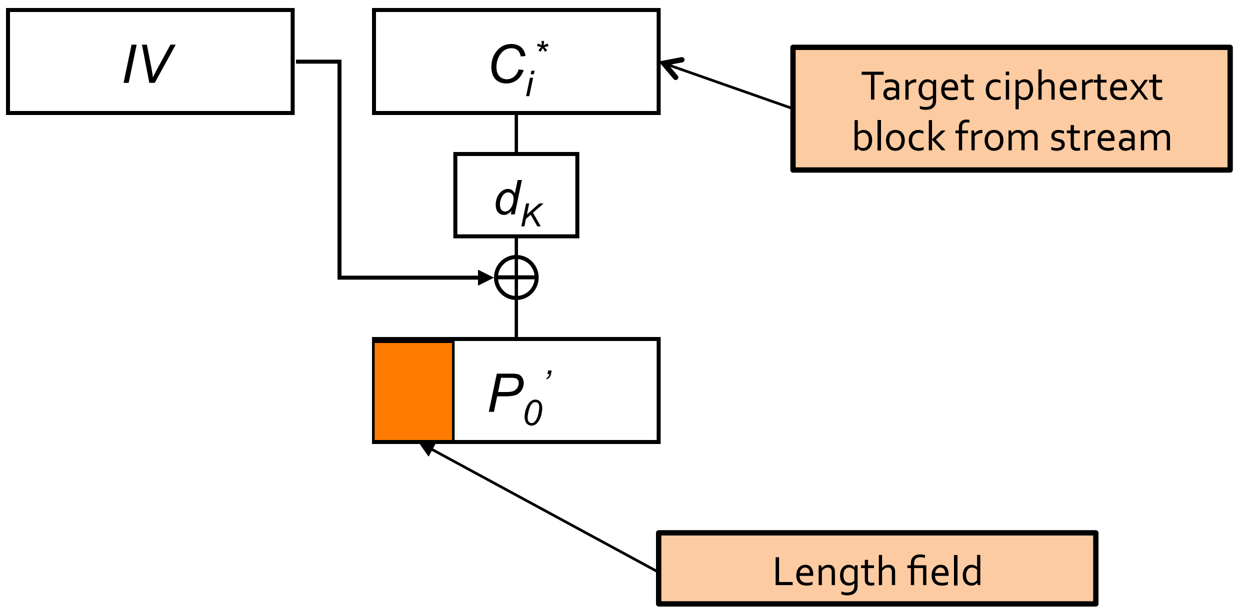
There's a significant semantic gap between AEAD's functionality and raw security guarantees, and all the things a developer expects a secure channel to provide

The SSH debacle



- Packet length field measures the size of the packet on the wire
  - Encrypted to hide the true length of SSH packets
- Needs random IV for CBC-mode to prevent chaining attack
- Construction with random IVs was proven IND-sfCCA secure (Bellare-Kohno-Namprempre, 2002)

Breaking SSH (Albrecht-Paterson-Watson, 2009)

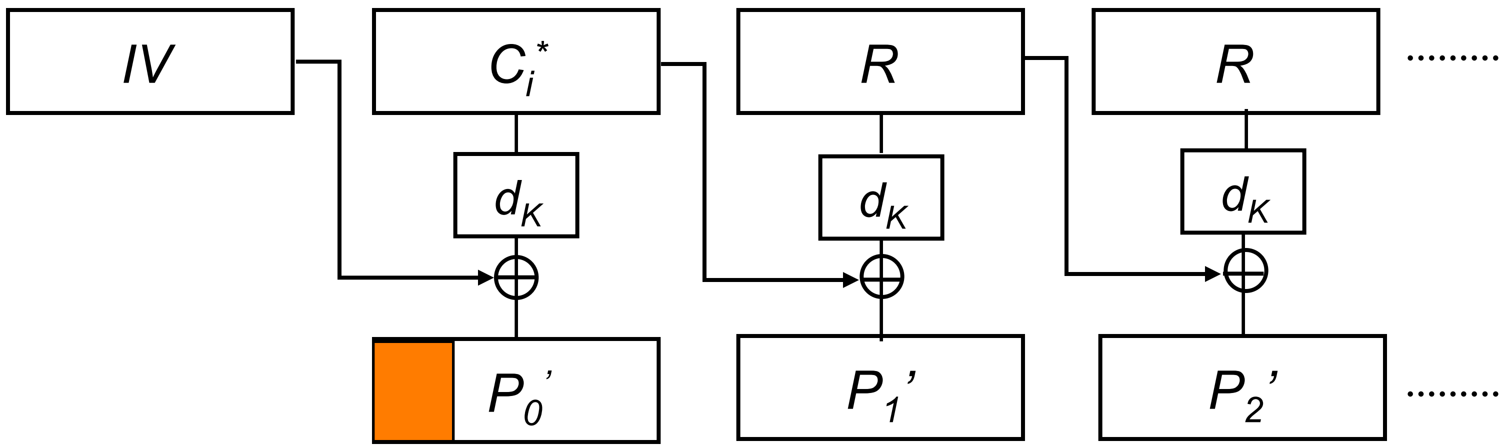


- The receiver will treat the first 32 bits of the calculated plaintext block as the packet length field for the new packet
- Here:

$$P_o' = IV \oplus d_K(C_i^*)$$

where *IV* is known

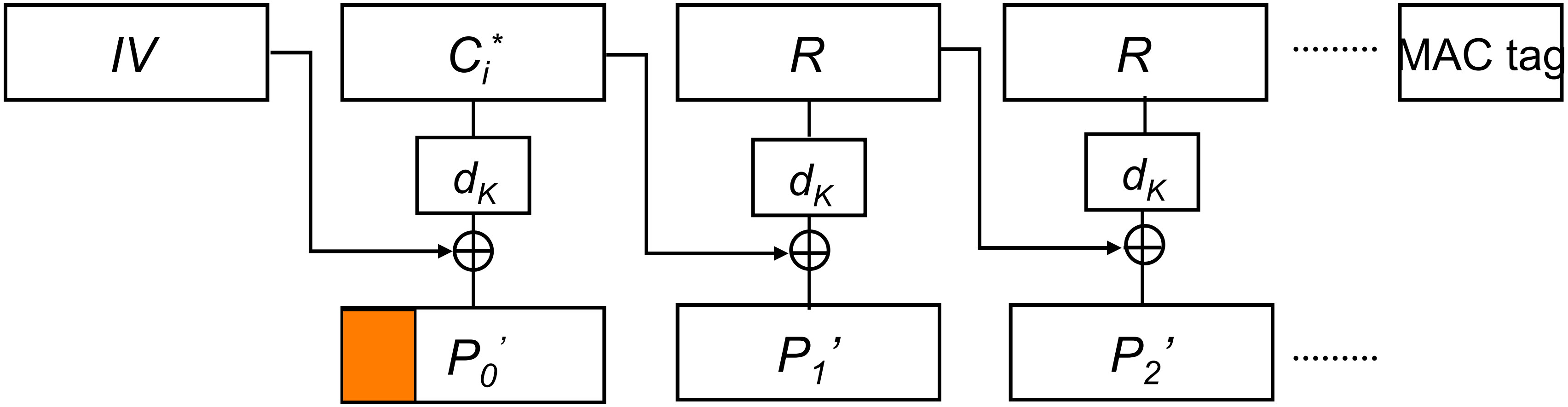
Breaking SSH (Albrecht-Paterson-Watson, 2009)



The attacker then feeds random blocks to the receiver

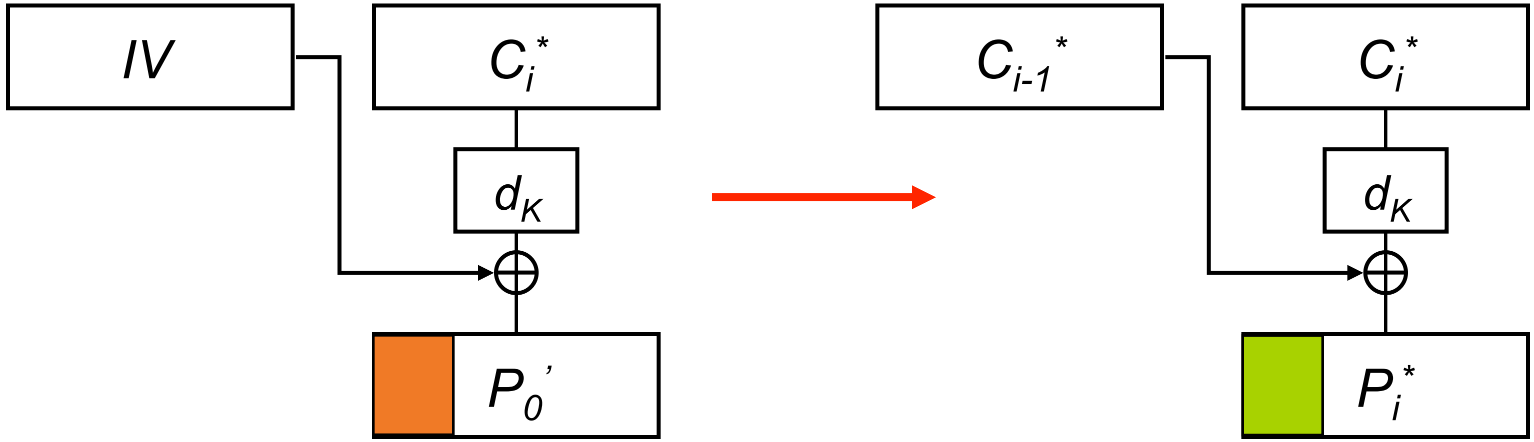
- One block at a time, waiting to see what happens at the server when each new block is processed
- This is possible because SSH runs over TCP and tries to do online processing of incoming blocks

Breaking SSH (Albrecht-Paterson-Watson, 2009)



- Once enough data has arrived, the receiver will receive what it thinks is the MAC tag
  - The MAC check will fail with overwhelming probability
  - Consequently the connection is terminated (with an error message)
- How much data is “enough” so that the receiver decides to check the MAC?
- Answer: whatever is specified in the length field

Breaking SSH (Albrecht-Paterson-Watson, 2009)



- Knowing IV and 32 bits of P_o', the attacker can now recover 32 bits of the target plaintext block:

$$P_i^* = C_{i-1}^* \oplus d_K(C_i^*) = C_{i-1}^* \oplus IV \oplus P_o'$$

- (Real attack is a bit more complicated, but follows this idea)

SSH debacle lessons

- Model used for security proof was inadequate
  - It assumed length known and atomic processing of ciphertexts
  - But fragmented adversarial delivery over TCP is possible
- Implementation can't know if complete ciphertext has arrived because of encrypted length field, unless it decrypts first block.
- That's not in any of the AE/AEAD security models!
- And there's no CAESAR requirement that looks like this!
- Modeling gap addressed in (Paterson-Watson, 2010) and (Boldyreva-Degabriele-Paterson-Stam, 2012)

Second example: cookie cutters

Bhargavan, Delignat-Lavaud, Fournet, Pironti, Strub 2014: cookie cutter attack on “HTTP over SSL/TLS”

- Attacker forces part of the HTTP header (e.g., cookie) to be cut off
- Partial message/header arrives and might be misinterpreted

Cookie cutters

Why doesn't this violate the proven integrity of SSL/TLS encryption?

```
6.2.1. Fragmentation

The record layer fragments information blocks
into TLSPlaintext records [...].  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, or a single
message MAY be fragmented across several
records).
```

RFC 5246 TLS v1.2

Cookie cutters

Why doesn't this violate the proven integrity of SSL/TLS encryption?

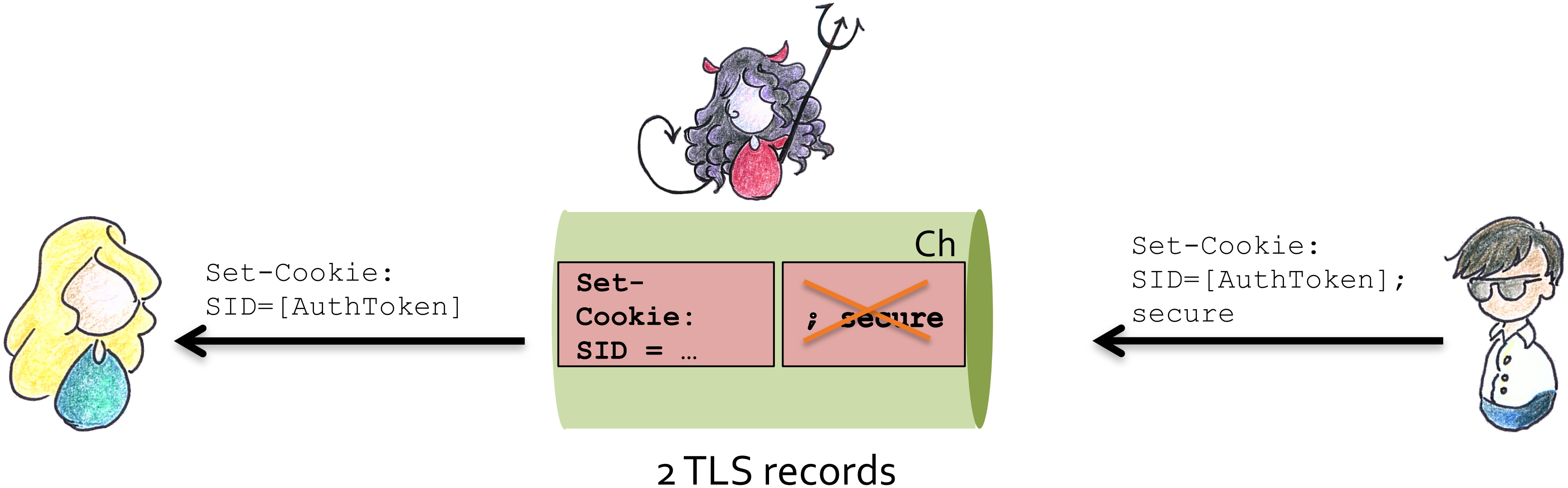
6.2.1. Fragmentation

The record layer fragments information blocks into TLSPlaintext records [...]. 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, **or a single message MAY be fragmented across several records**).

RFC 5246 TLS v1.2

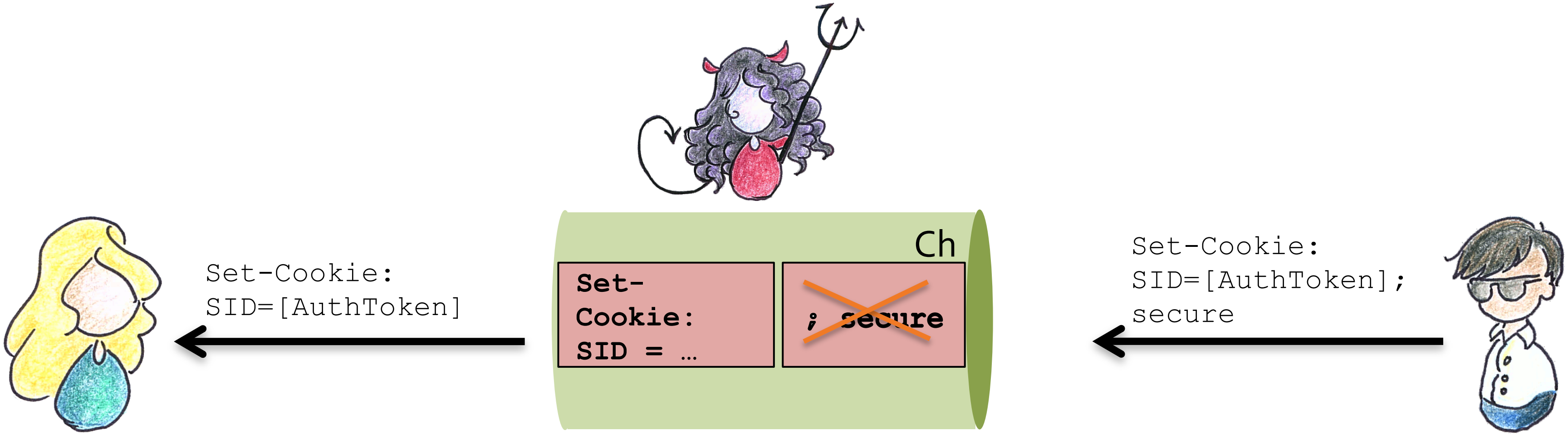
Cookie cutters

- So SSL/TLS can (and will) fragment when *sending*
- Compare to SSH that has to deal with fragments when *receiving*
- Both protocols provide a *streaming* interface to applications, not a message-oriented one

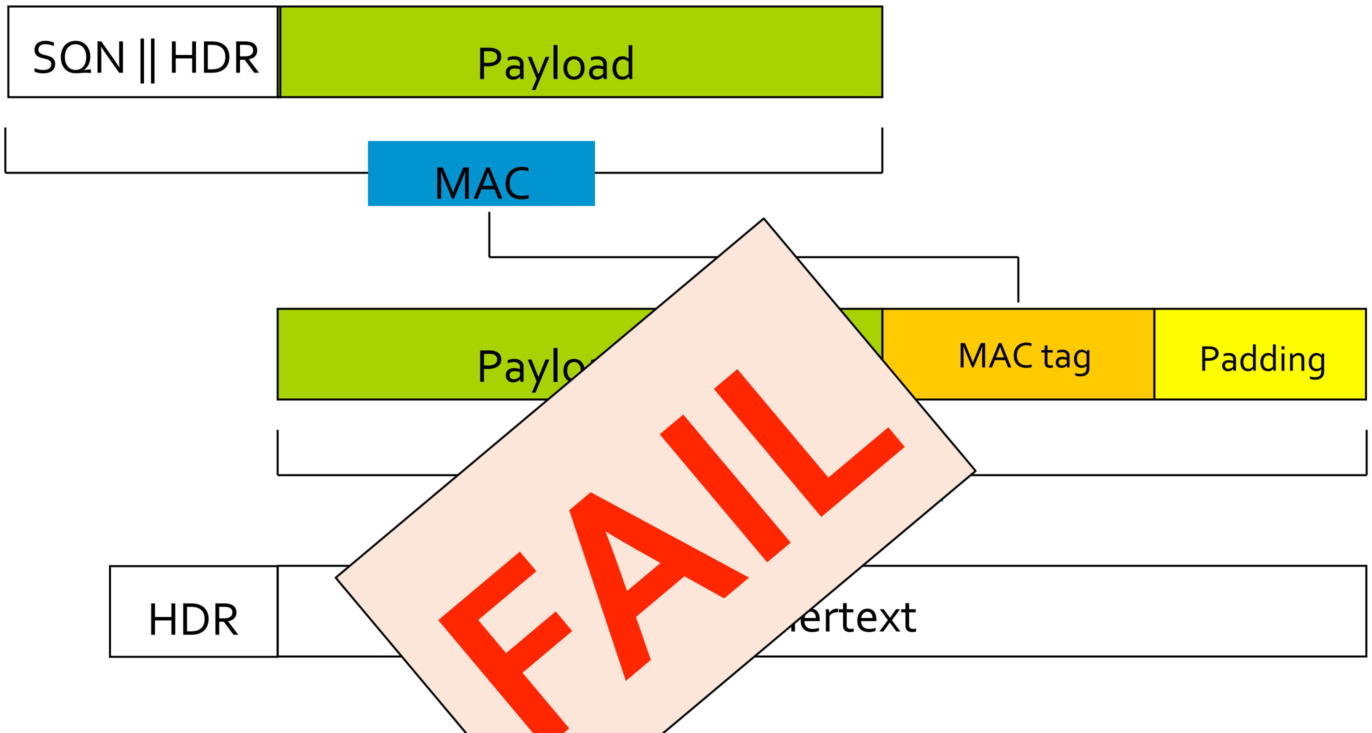


Cookie cutters

- It's up to the calling application to deal with message boundaries if it wants to use SSL/TLS for atomic message delivery
- Cookie cutter attack relies on a buggy browser that does not check for correct HTTP message termination
- This happens in practice, presumably because developers do not understand the interface provided by SSL/TLS



TLS Record Protocol: MAC-Encode-Encrypt (MEE)



MAC HMAC-MD5, [illegible]AC-SHA1, HMAC-SHA256

Encrypt CBC-AES128, CBC-AES256, CBC-3DES, RC4-128

From AEAD to secure channels

From AEAD to secure channels

- SSL/TLS is not alone in presenting a streaming interface to applications
- Also SSH "tunnel mode", QUIC
- What security can we hope for from such a channel?
- Boldyreva-Degabriele-Paterson- Stam (2012) already treated the case where the receiver handles fragmented ciphertexts
- In Fischlin-Günther-Marson-Paterson (2015), we provide a systematic study of the case where *both* sender and receiver may fragment, as in TLS

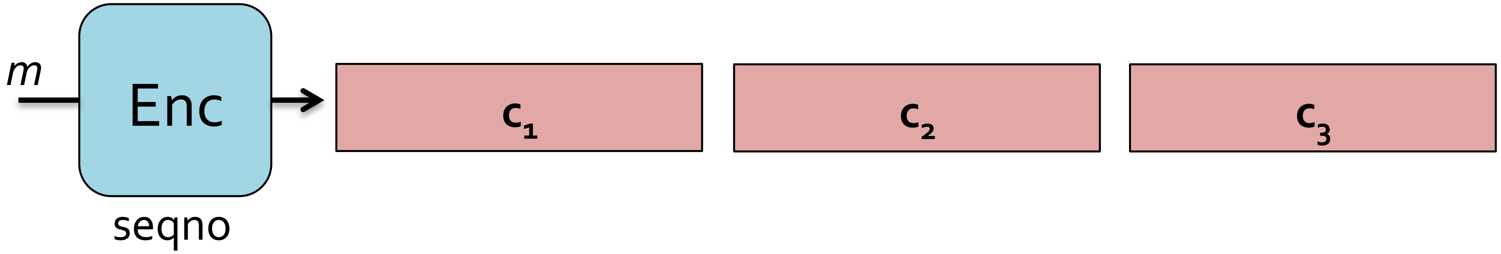
Streaming secure channels (FGMP15)

- Defining CCA and integrity notions in the full streaming setting is non-trivial!
  - Hard part is to define when adversary's decryption queries deviate from sent stream, and from which point on to suppress decryption oracle outputs
- We develop streaming analogues of IND-CPA, IND-CCA, INT-PTXT and INT-CTXT
- We recover an analogue of the classic relation

IND-CPA + INT-CTXT ➔ IND-CCA

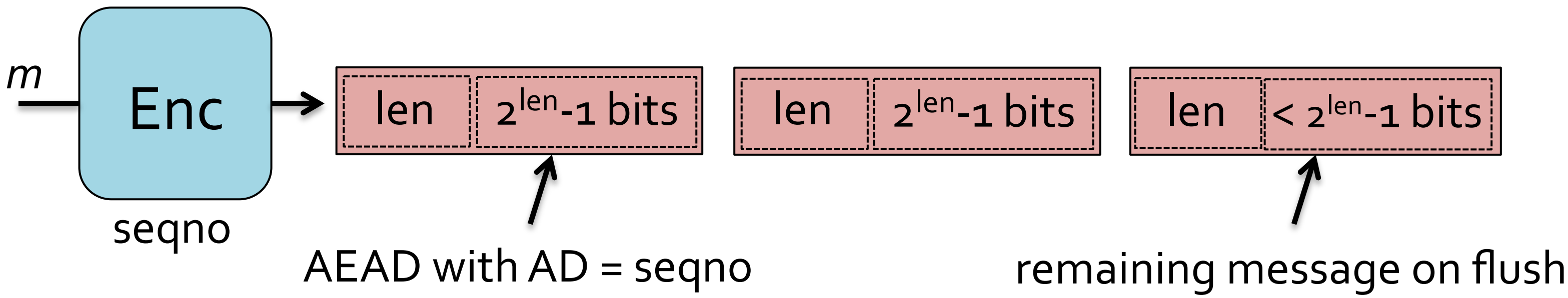
Streaming secure channels (FGMP15)

- We give a generic construction for a secure streaming channel that validates the SSL/TLS design
- The construction uses AEAD as a component
- Security as streaming channel follows from standard AEAD security properties



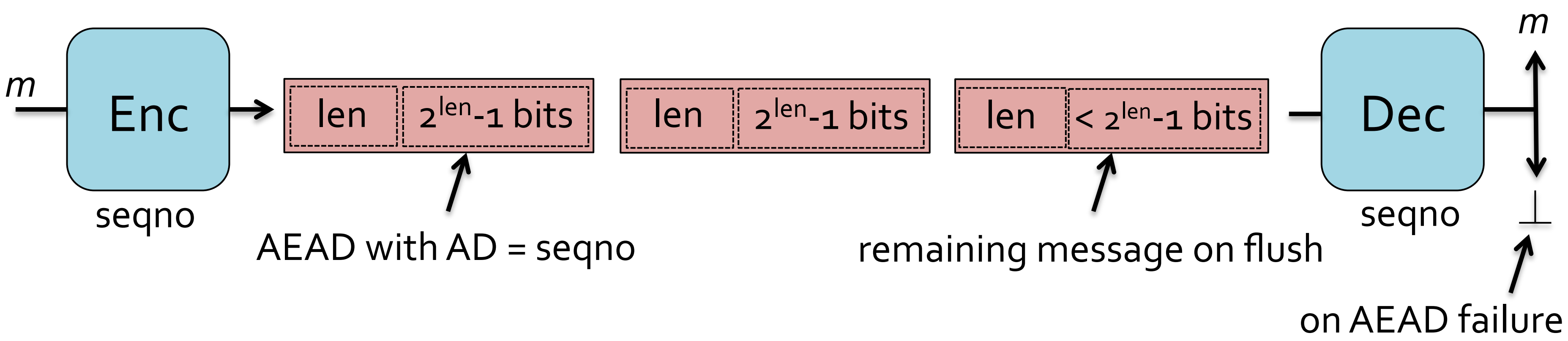
Streaming secure channels (FGMP15)

- We give a generic construction for a secure streaming channel that validates the SSL/TLS design
- The construction uses AEAD as a component
- Security as streaming channel follows from standard AEAD security properties



Streaming secure channels (FGMP15)

- We give a generic construction for a secure streaming channel that validates the SSL/TLS design
- The construction uses AEAD as a component
- Security as streaming channel follows from standard AEAD security properties



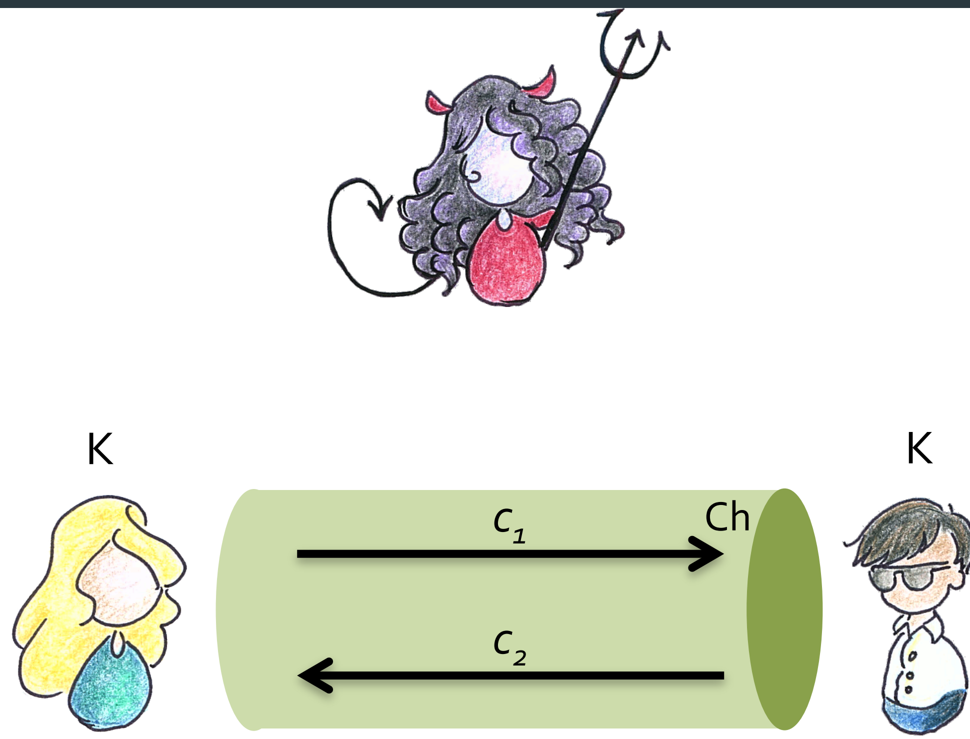
- TLS 1.3 has unsent seqno as AD and sent but unprotected length, but also sent + protected version number and content type fields

Are we there yet?

Are we there yet?

$c_1 = \mathrm{Enc}_K(m_1)$

$m_2 = \mathrm{Dec}_K(c_2)$



$m_1 = \mathrm{Dec}_K(c_1)$

$c_2 = \mathrm{Enc}_K(m_2)$

Context

GCHQ
MASTERING THE INTERNET (MTI)

The Snowden revelations tell us that **mass surveillance** of Internet traffic **is** taking place.

Just encrypting traffic is not enough to prevent mass surveillance.

- Backdoors in cryptographic standards (e.g. NIST Dual EC DBRG).
- Extraction of server keys by legal means or by penetration of target.
- Active attacks on PKI (certificate substitution).
- Backdoors in cryptographic software, exploiting timing, covert channels, ….
- Other means as yet unknown.

Algorithm Substitution Attacks

Bellare-Paterson-Rogaway (2014):

What "other means" are possible for carrying out mass surveillance against encrypted traffic , and what can we do to protect against them?

Our focus was narrow but carefully chosen:

Algorithm Substitution Attacks (ASAs) on **Symmetric Encryption** (SE).

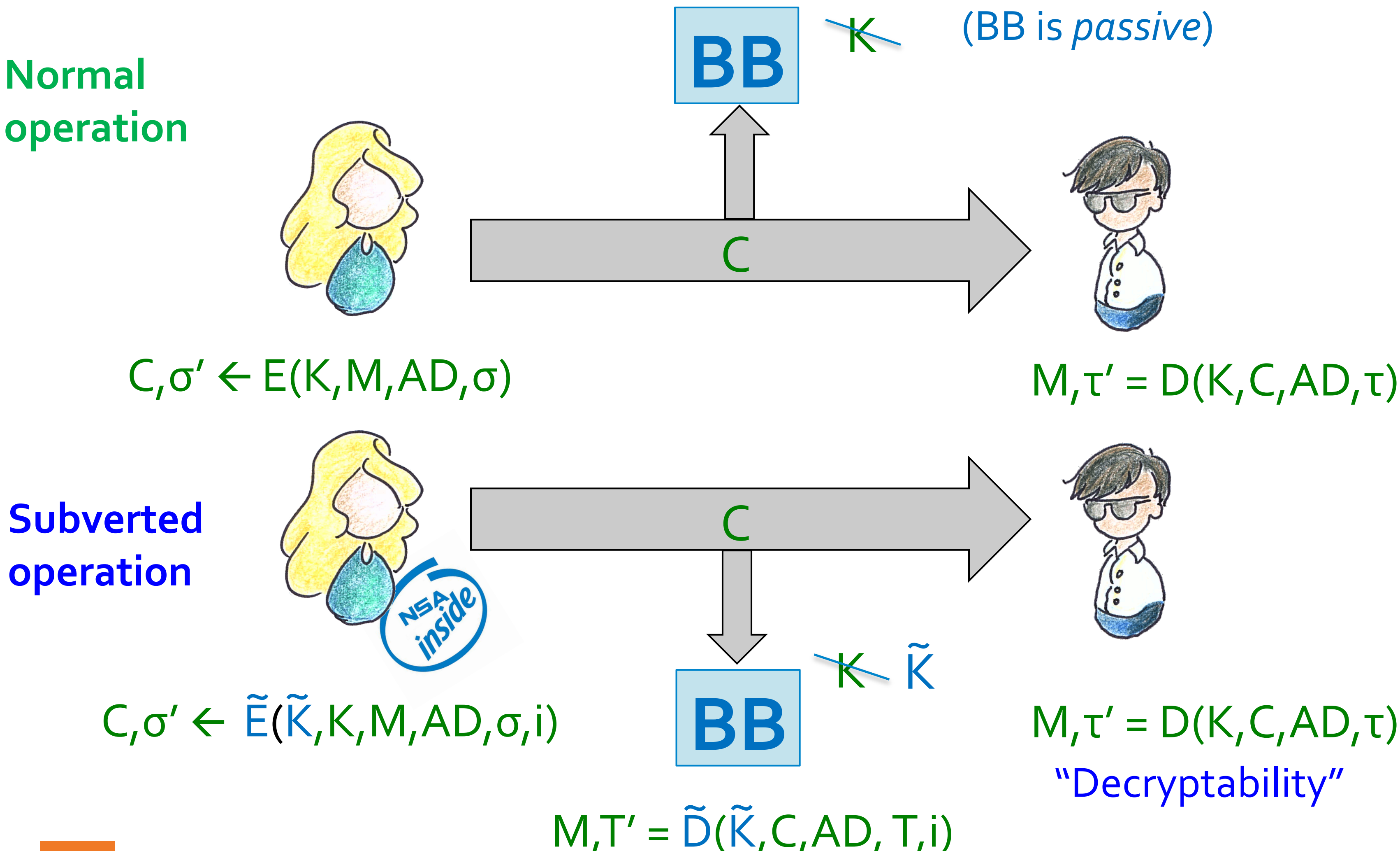
Basic idea of ASAs:

- Big Brother Adversary substitutes real encryption algorithm E with subverted one $\tilde{E}$.
- Ciphertexts generated by E and $\tilde{E}$ look the same to ordinary users.
- But ciphertexts generated by $\tilde{E}$ leak everything to Big Brother.

ASAs are a little-explored but realistic means of enabling mass surveillance.

- Informal treatments: Young-Yung (1996, 1997), Goh, Boneh, Pinkas, Golle (2003)

The setting for ASAs against Symmetric Encryption



Where do ASAs make sense?

- Closed-source software.
- Complex, open-source software not subject to sufficient scrutiny (cf Heartbleed bug in OpenSSL).
- Backdoor in compiler an mount ASA at compile time (as per Ken Thompson's "Reflections on Trusting Trust" paper).
- Hardware implementation, especially tamper-evident/ proof.

Modelling ASAs and security against them

- We gave **formal definitions** for SE secure against ASAs, using **two** adversaries:

  Detection Adversary: models ordinary users in possession of K but not $\tilde{K}$, who wants to know if an ASA is underway.

  Surveillance Adversary: models Big Brother in possession of $\tilde{K}$ but not K, who wants to read all users' traffic.

- **Security against ASAs**:

  Either Detection Adversary trivially succeeds

  OR

  Surveillance Adversary miserably fails.

ASAs against randomised schemes

- Any **randomised, stateless** SE scheme is vulnerable to an undetectable ASA allowing Big Brother to efficiently recover the encryption key K.
- Basic idea:
  - Let F:{0,1}*-> {0,1} be a PRF with key $\tilde{K}$.
  - To leak K[j], bit j of key K, algorithm $\tilde{E}$ repeatedly encrypts using E and fresh, random coins to produce C such that F($\tilde{K}$,C,j) = K[j] .
  - BB is equipped with $\tilde{K}$ so can efficiently recover bit K[j] from C.
  - User does not know $\tilde{K}$ so cannot distinguish C from normal ciphertexts.
  - (Additional complexity needed to deal with different indices j and different keys K_i.)
- Attack extends to randomised, stateful setting too.

Defeating ASAs

- Stateless, deterministic schemes can't achieve semantic security.
- But randomised schemes are now bad.
  - Runs counter to our received wisdom on how to achieve semantic security!
- We make use of a class of stateful, deterministic schemes, namely **unique ciphertext** schemes:

For any key K, message M, associated data AD, and state τ, there is at most one ciphertext C that decrypts to M under K.

Defeating ASAs

Theorem:

Let $\Pi = (K,E,D)$ be a unique ciphertext scheme. Let $\tilde{\Pi} = (\tilde{K},\tilde{E},\tilde{D})$ be any subversion of Π that is decryptable*.

Then any surveillance adversary B against Π has zero advantage.

*ciphertexts generated by $\tilde{E}$ decrypt correctly under D.

Defeating ASAs

- The preceding theorem is easy to instantiate.
- For example, start with a nonce-based symmetric encryption scheme that is *tidy* in the sense of [NRS14].
- Set the nonce N to be a counter in both E and D to make a doubly stateful scheme.
- Easy to show that this scheme has unique ciphertexts.

The role of decryptability

- We have presented decryptability as a natural, minimal condition required to make BB adversary undetectable.
- Decryptability and undetectability are actually incomparable notions.
- If subversions $\tilde{E}$ are allowed to deviate from decryptability, then it is easy to design an undetectable $\tilde{E}$ that leaks K.
  - Special trigger message m^* outputs K as ciphertext.
- And it may be hard to distinguish for the communicating partner to distinguish such deviations from communications errors.
- Degabriele-Farshim-Poettering (2015) investigate and resolve this issue.

Closing remarks

Closing remarks

- We've seen the evolution from simple security models for symmetric encryption to more sophisticated security notions for secure channels
- Yet the relevant part of the cryptography community is mostly focussed on AEAD and CAESER
- Key take-away: think top-down, not bottom-up (from API to crypto, not the reverse)
- The post-Snowden adversary brings new and interesting research challenges

Closing remarks