

Secure Channels – Are We There Yet?

Kenny Paterson

Information Security Group

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



ROYAL
HOLLOWAY
UNIVERSITY
OF LONDON



Motivation

Why do we still need research on secure channels?

- Secure communications is still the most common real-world application of cryptography today.
 - SSL/TLS, DTLS, IPsec, SSH, OpenVPN,...
 - WEP/WPA/WPA2
 - GSM/UMTS/4g/LTE
 - Cryptocat, OTR, SilentCircle, OpenPGP, iMessage, Telegram, Signal, and a thousand other messaging apps
 - QUIC, MinimalT, TCPcrypt

Why do we still need research on secure channels?

- Secure communications is still the most common real-world application of cryptography today.
 - *SSL/TLS, DTLS, IPsec, SSH, OpenVPN,...*
 - *WEP/WPA/WPA2*
 - *GSM/UMTS/4g/LTE*
 - *Cryptocat, OTR, SilentCircle, OpenPGP, iMessage, Telegram, Signal, and a thousand other messaging apps*
 - *QUIC, MinimalT, TCPcrypt*
- **Bottom line: it might be boring, but we keep getting this wrong, and it's not clear we're getting any better at it.**

Overview

- Secure channels and their properties
- AEAD
- AEAD \neq secure channel
 - SSH and TLS examples
- Building better models
- Closing remarks



Secure channels and their properties

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.
- **Detection of deletion**
 - Detection that messages have been deleted by the adversary or dropped by the network.
- **Detection of re-ordering**
 - Ensuring that the relative order of messages in *each* direction on the secure channel is preserved.
 - Possibly re-ordering the event of violation.
- **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 black-box for delivering messages securely.

Additional API-driven requirements

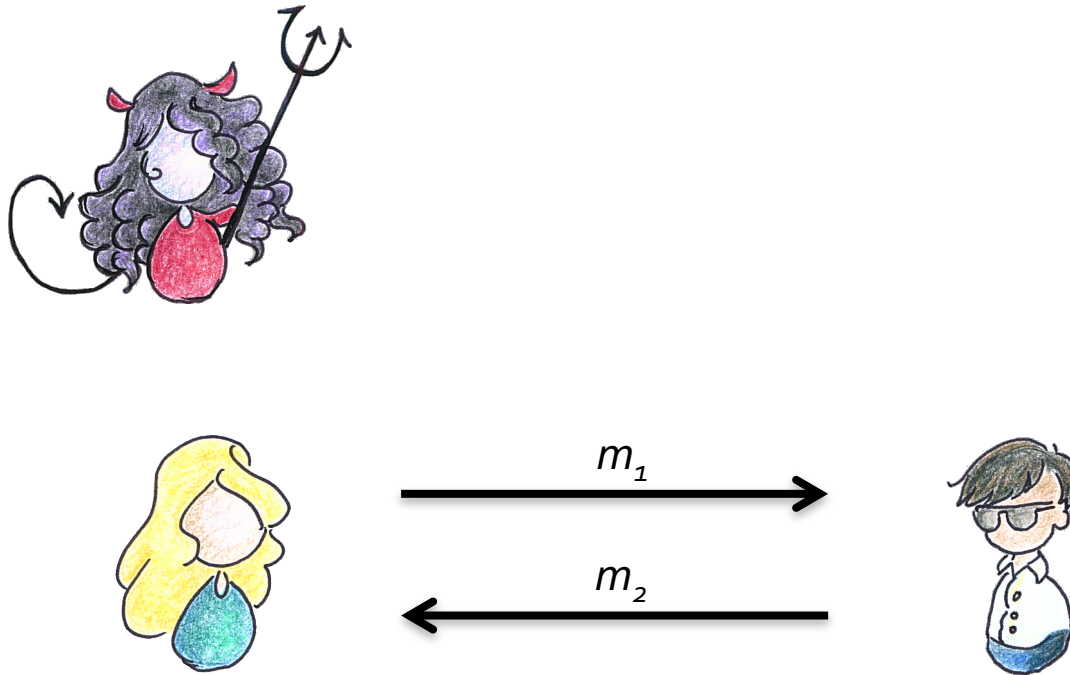
- Does the channel provide a stream-based functionality or a message-oriented functionality?
- Does the channel accept messages of arbitrary length and perform its own fragmentation and reassembly, or is there a maximum message length?
- How is error handling performed? Is a single error fatal, leading 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 the basic security definitions for symmetric encryption**



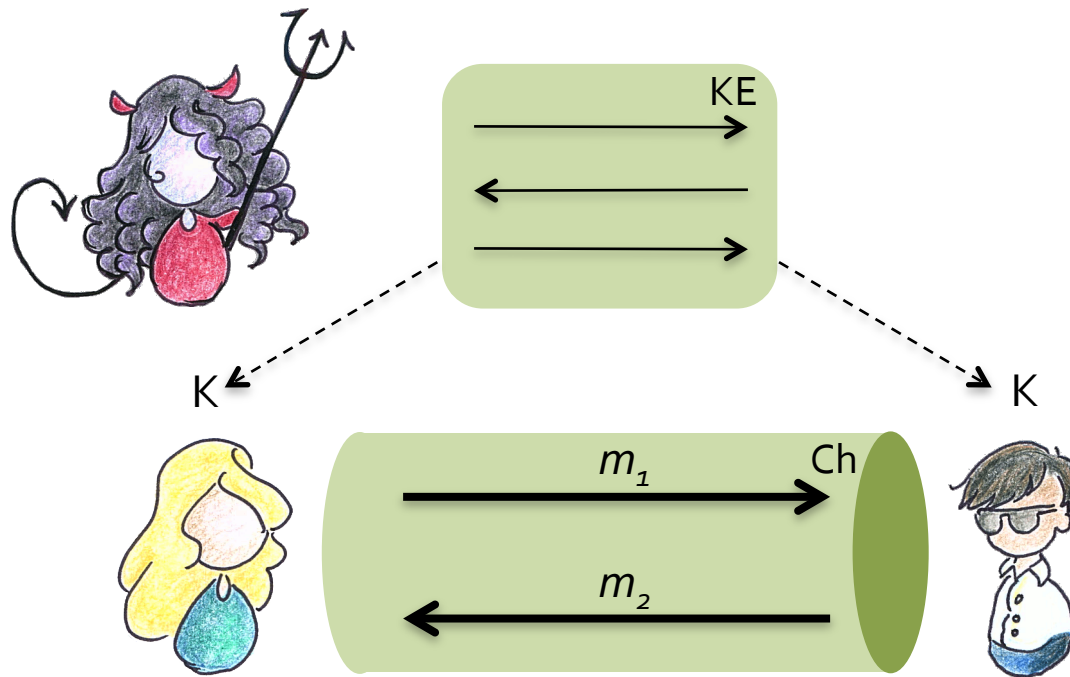
AEAD



Security for Symmetric Encryption



Security for Symmetric Encryption

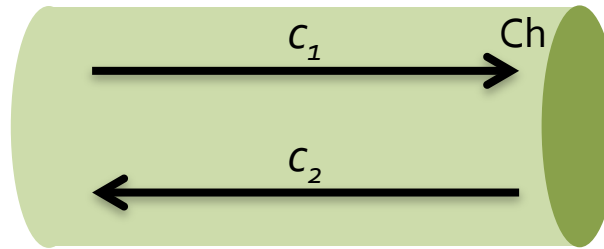
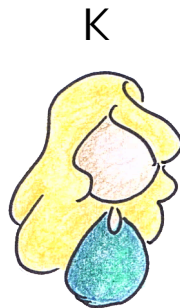


Security for Symmetric Encryption



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

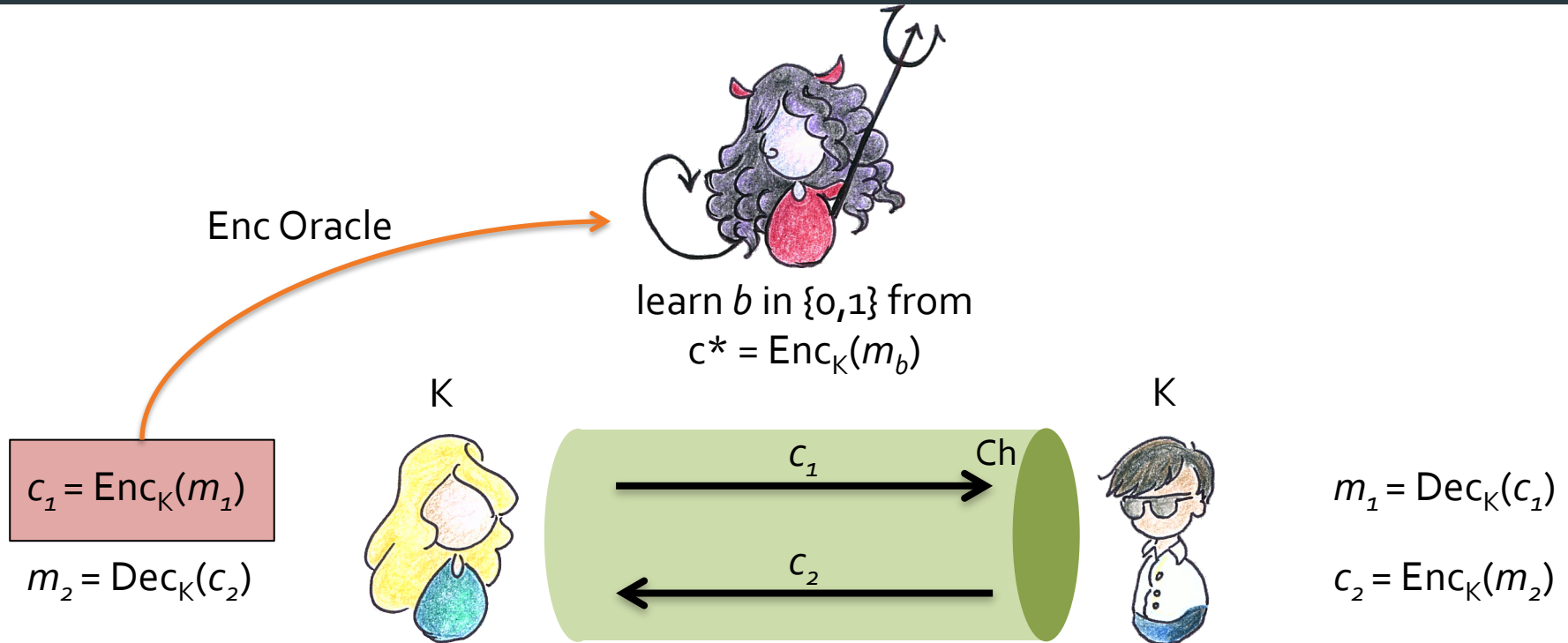
$$m_2 = \text{Dec}_K(c_2)$$



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

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

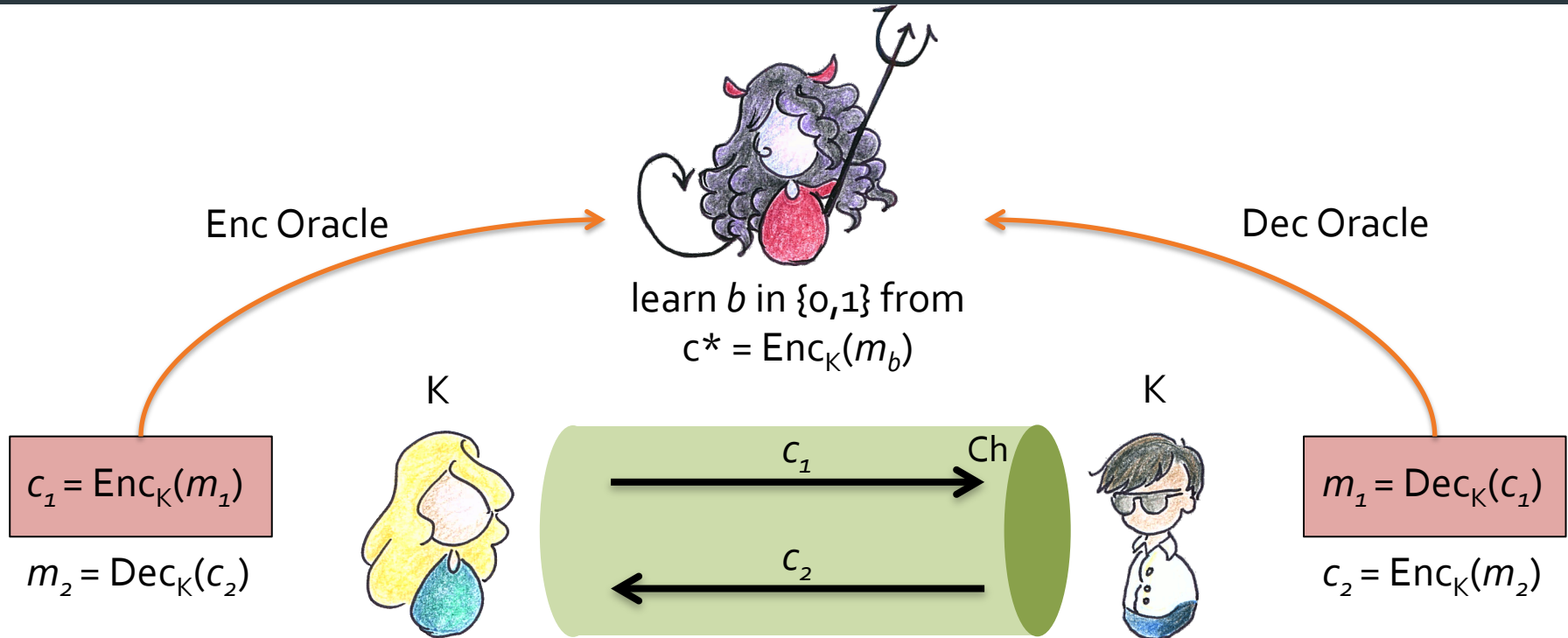
Security for Symmetric Encryption – Confidentiality



IND-CPA

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

Security for Symmetric Encryption – Confidentiality



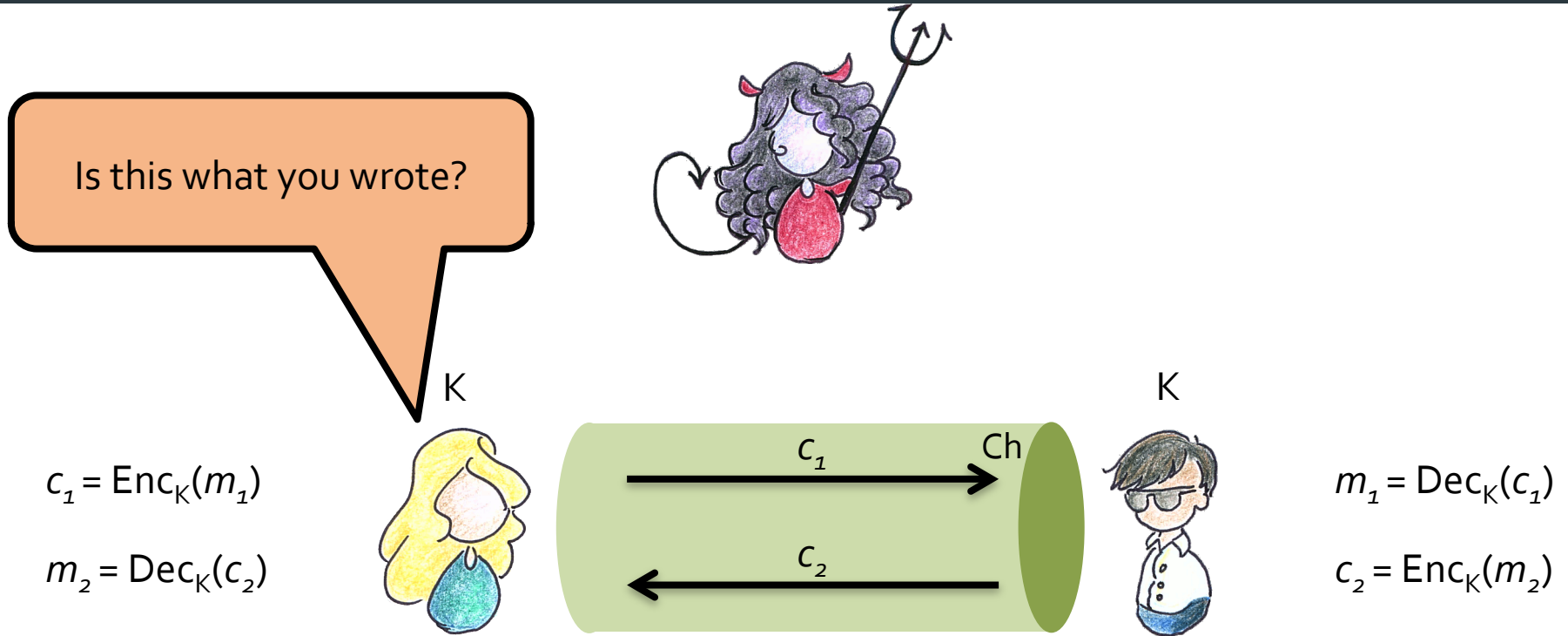
IND-CPA

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

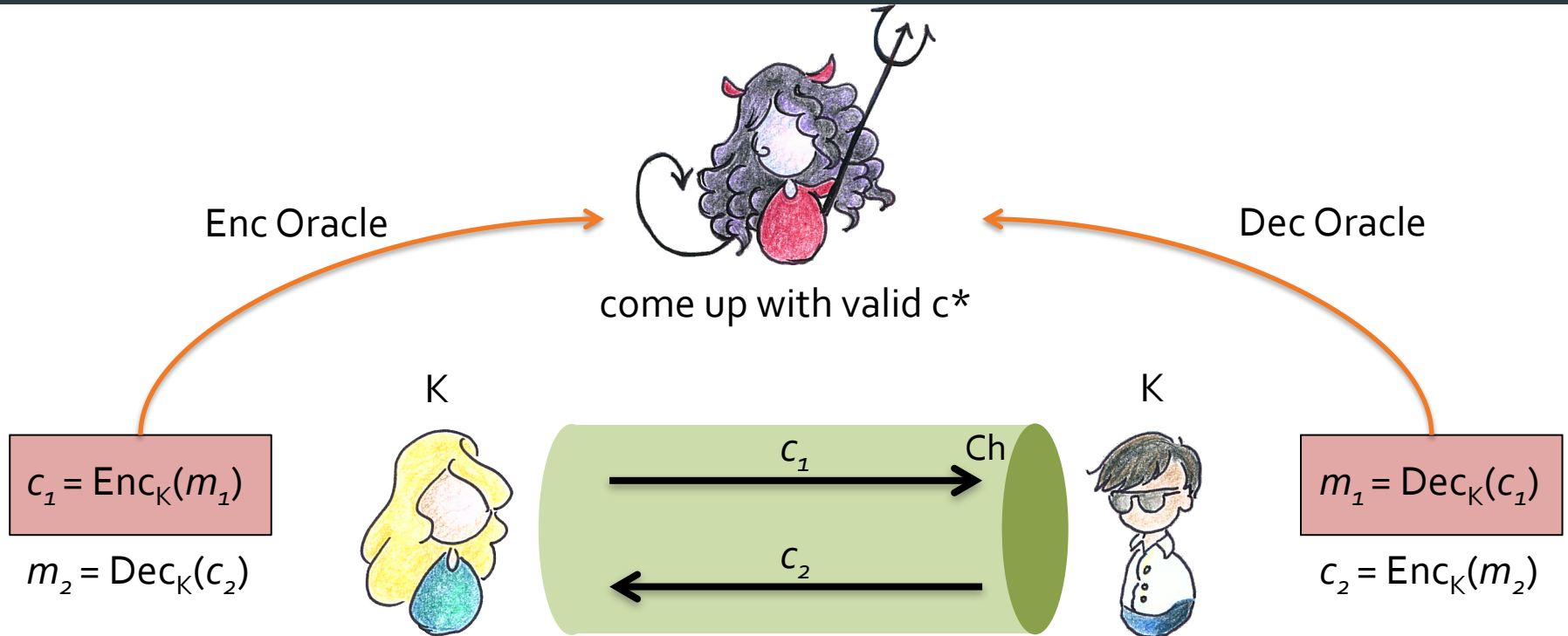
IND-CCA

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

Security for Symmetric Encryption – Integrity

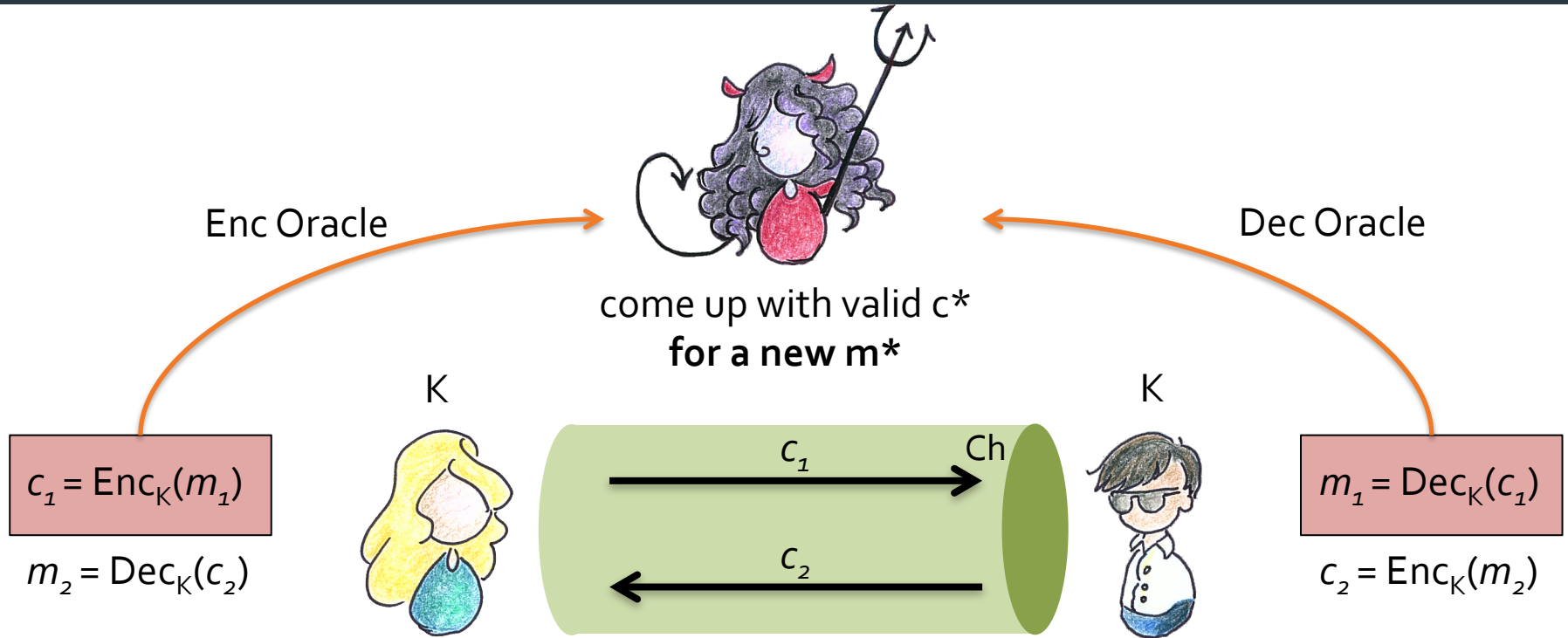


Security for Symmetric Encryption – Integrity



INT-CTXT
(Bellare, Rogaway, 2000)

Security for Symmetric Encryption – Integrity



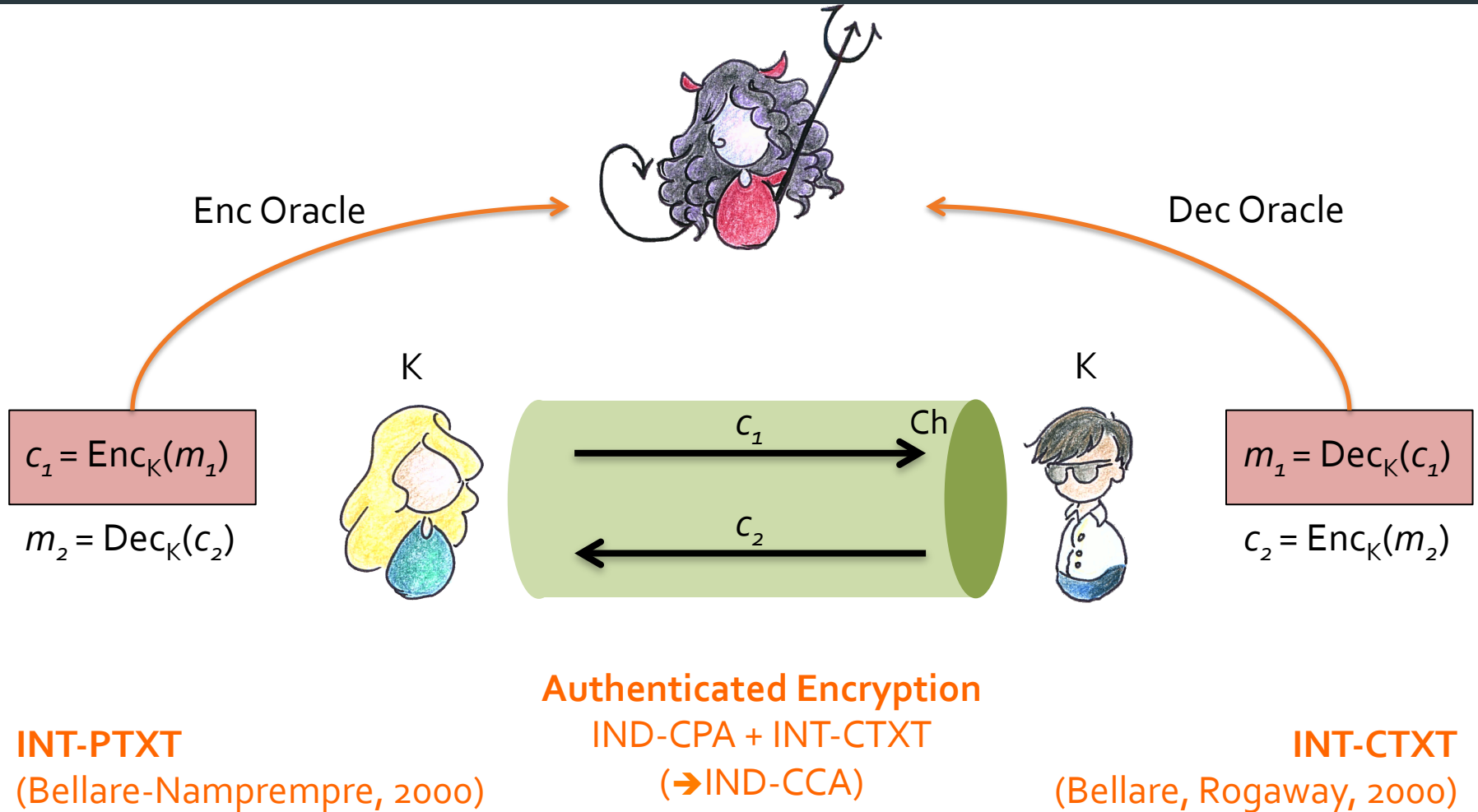
INT-PTXT

(Bellare-Namprempe, 2000)

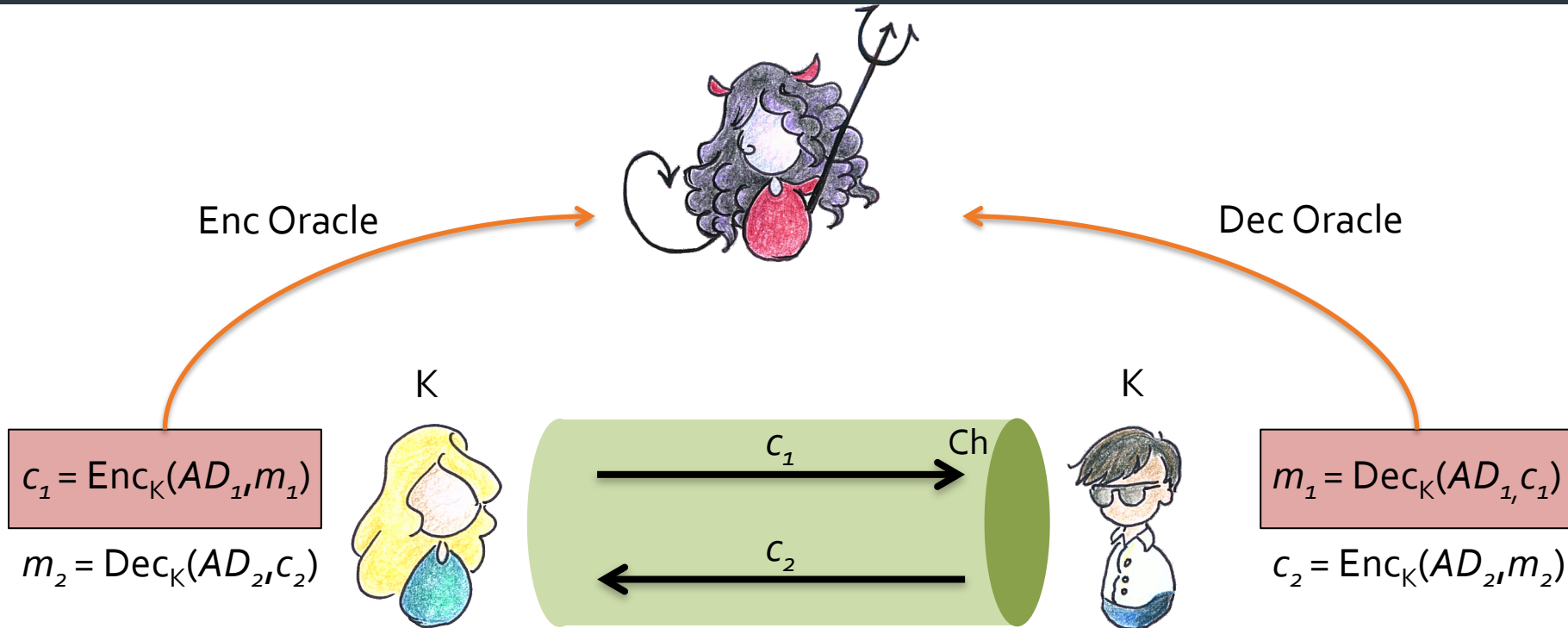
INT-CTXT

(Bellare, Rogaway, 2000)

Security for Symmetric Encryption – AE



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

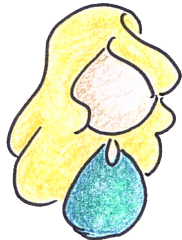
Which came first?

K

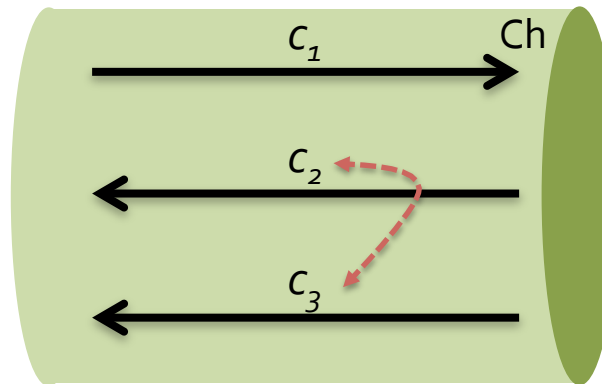
$$c_1 = \text{Enc}_K(AD_1, m_1)$$

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

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



K

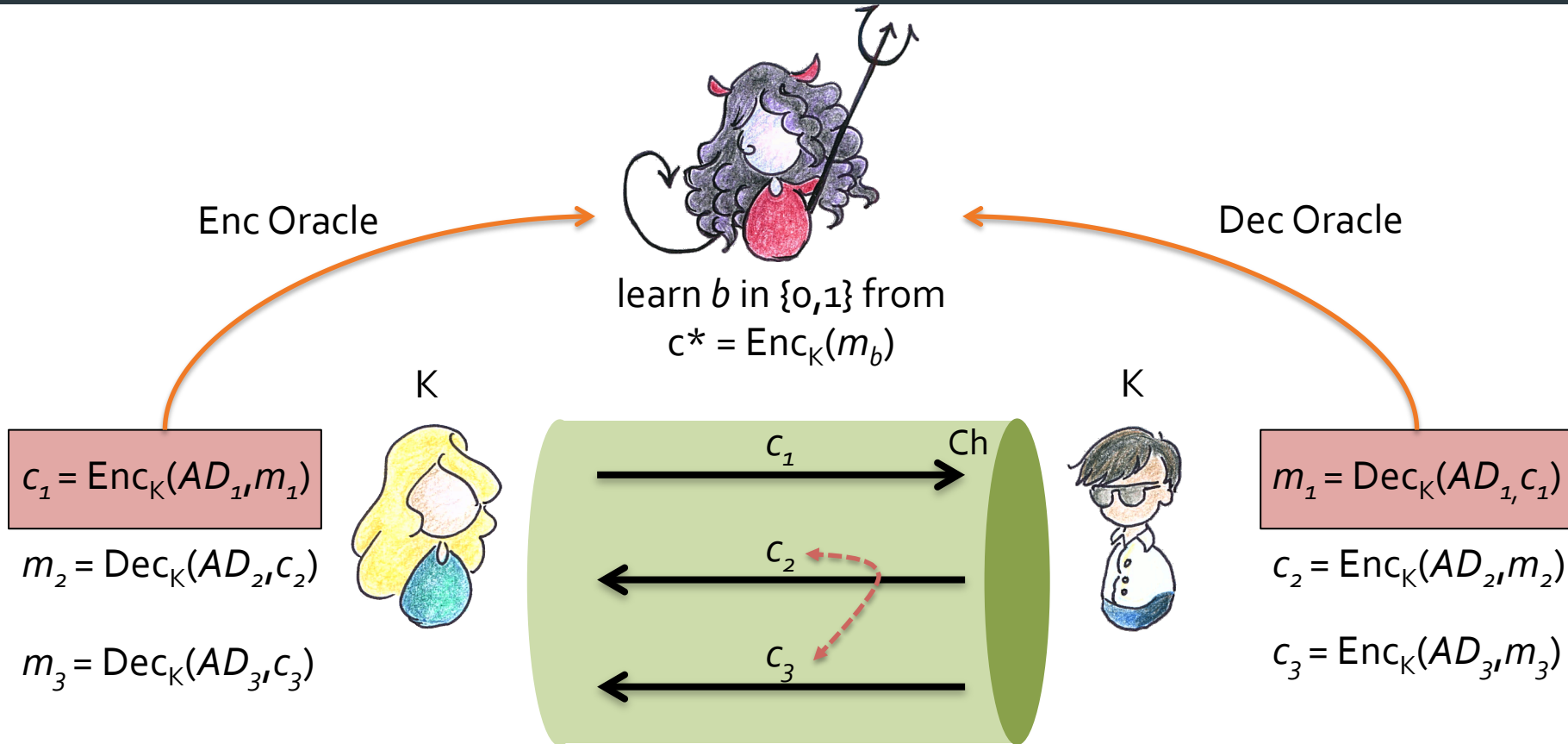


$$m_1 = \text{Dec}_K(AD_1, c_1)$$

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

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

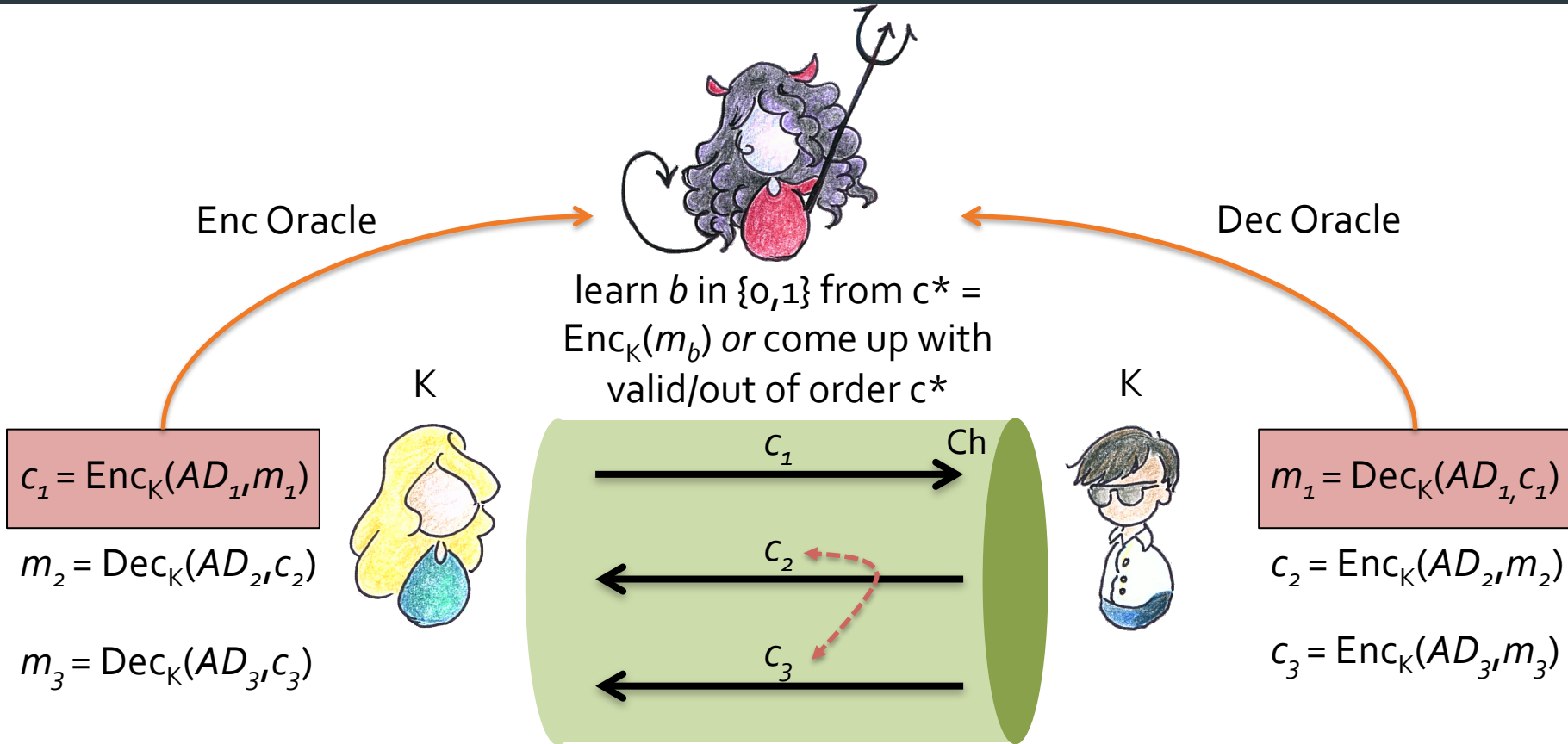
Security for Symmetric Encryption – stateful AEAD



IND-sfCCA

(Bellare-Kohno-Namprempre, 2002)

Security for Symmetric Encryption – stateful AEAD



IND-sfCCA

Stateful AEAD

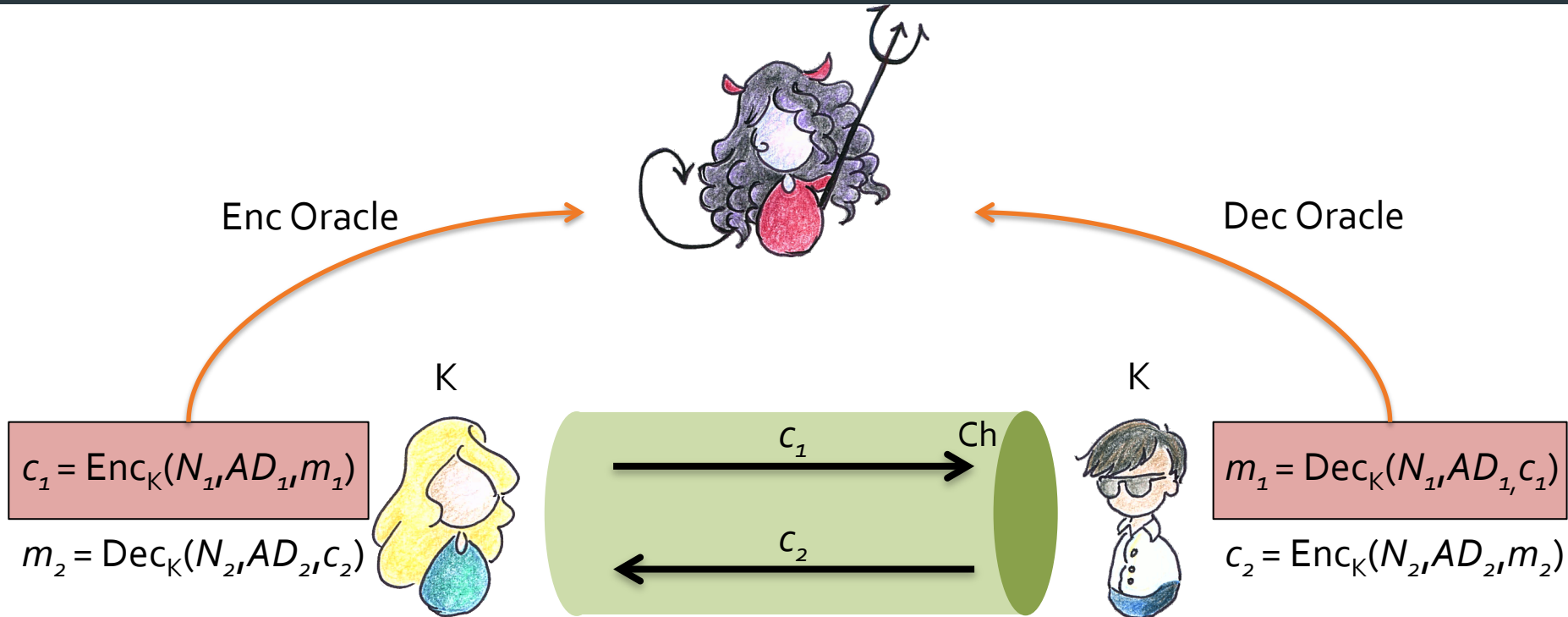
INT-sfCTXT

(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)

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**.
- CAESAR has involved dozens of person-years of effort and led to a major uptick in research activity.
- **It seems that most of the cryptographic community has settled on nonce-based AEAD as their design target.**



AEAD \neq secure channel

AEAD \neq secure channel

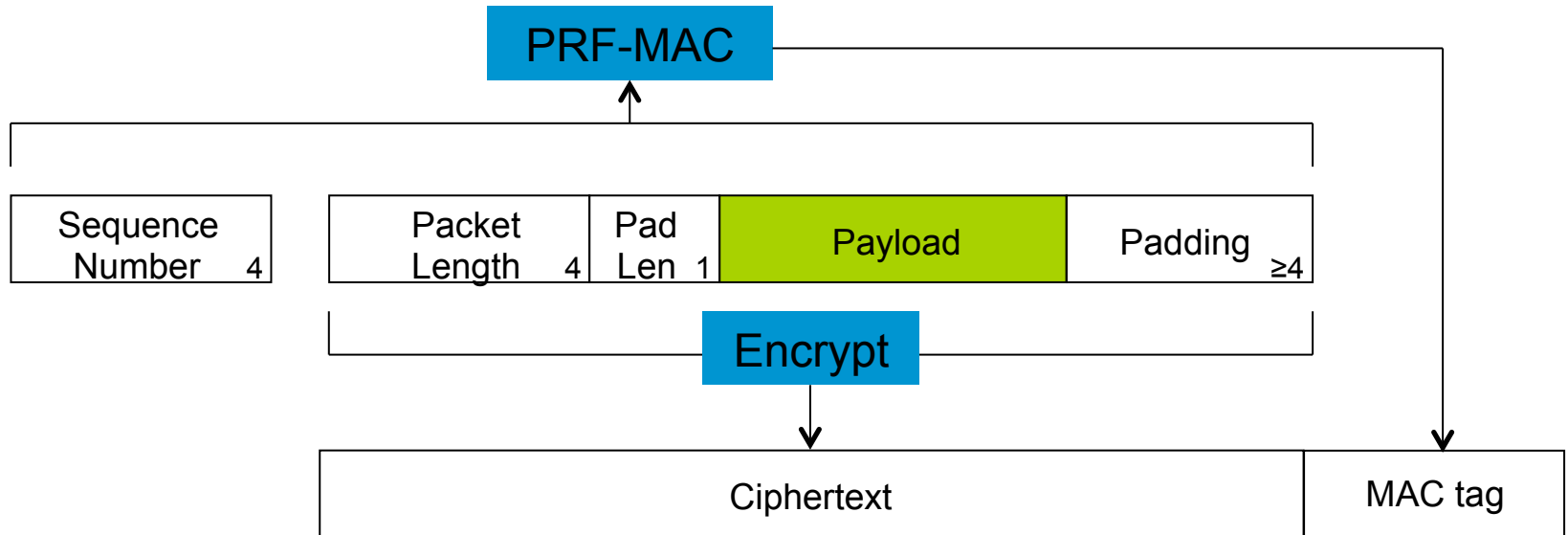
- Recall our application developer:
 - He wants a drop-in replacement for TCP that's secure.
 - Actually, he might *just* want to send and receive some atomic messages and not a TCP-like stream.
- To what extent does AEAD meet these requirements?
- It doesn't...

AEAD \neq secure channel



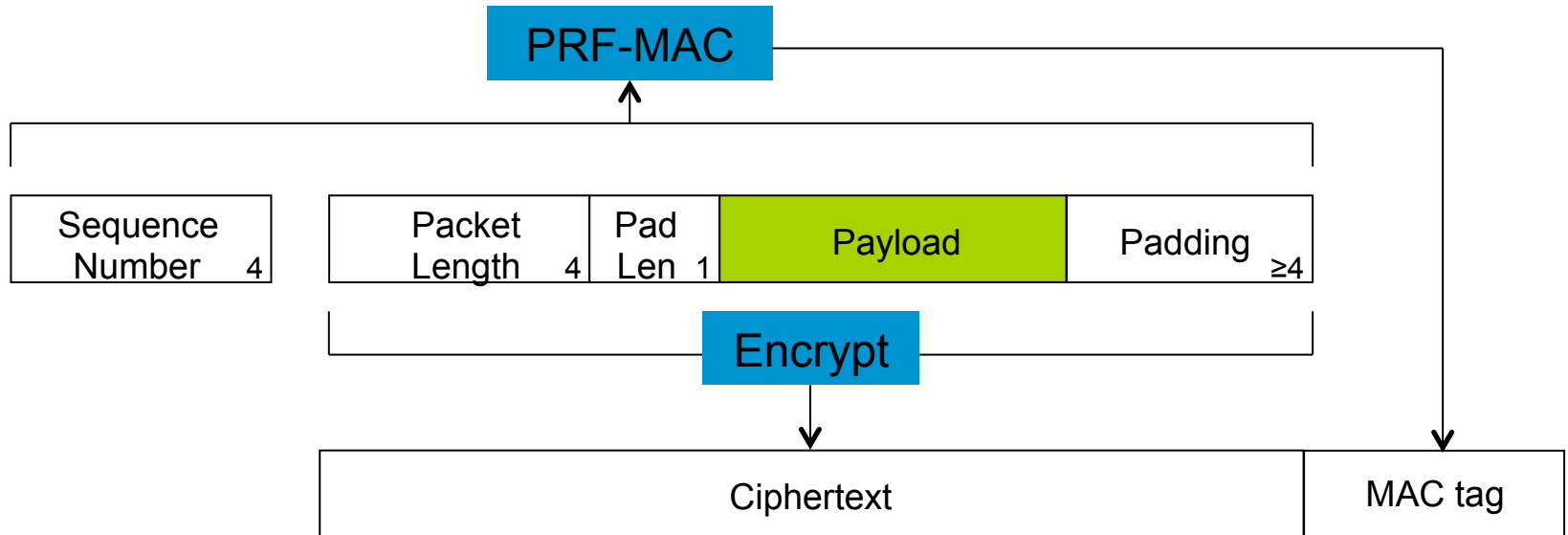
There's a significant semantic gap between AEAD's functionality and raw security guarantees, and the things a developer expects a secure channel to provide.

First example: SSH Binary Packet Protocol (RFC 4253)



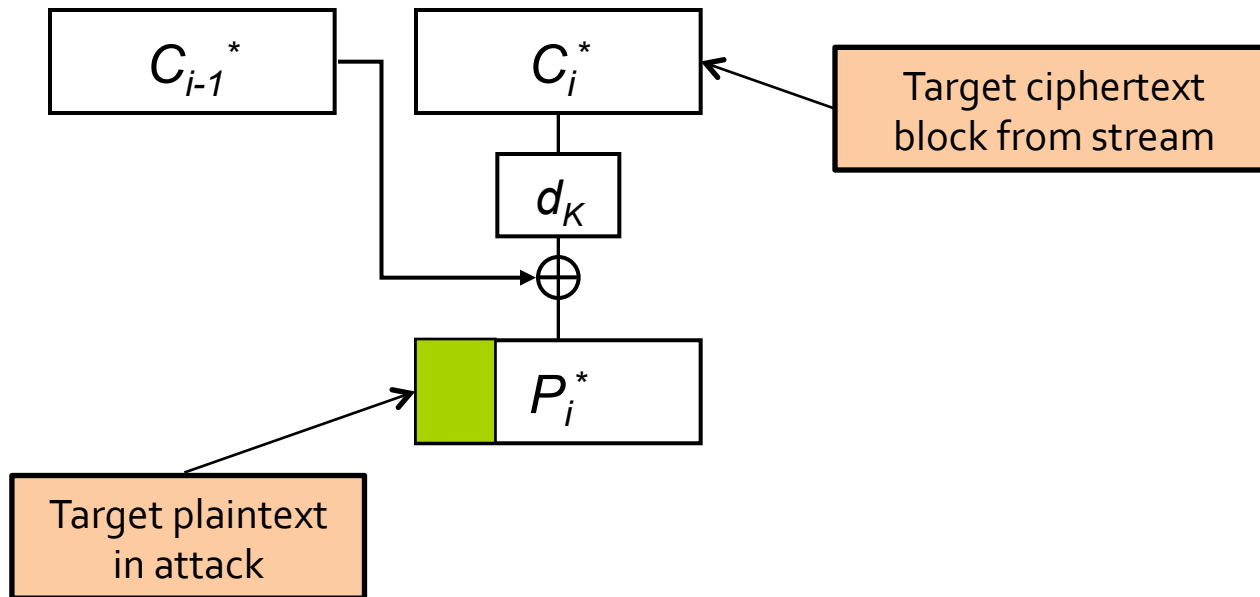
- Encode-then-E&M construction, stateful because of inclusion of 4-byte sequence number.
- Packet length field measures the size of the packet: $|\text{PadLen}| + |\text{Payload}| + |\text{Padding}|$.
 - Encrypted, so sequence of encrypted packets looks like a long string of random bytes.
- Encryption options in RFC 4253: CBC mode; RC4.
- AES-CTR defined in RFC 4344.

First example: SSH Binary Packet Protocol (RFC 4253)

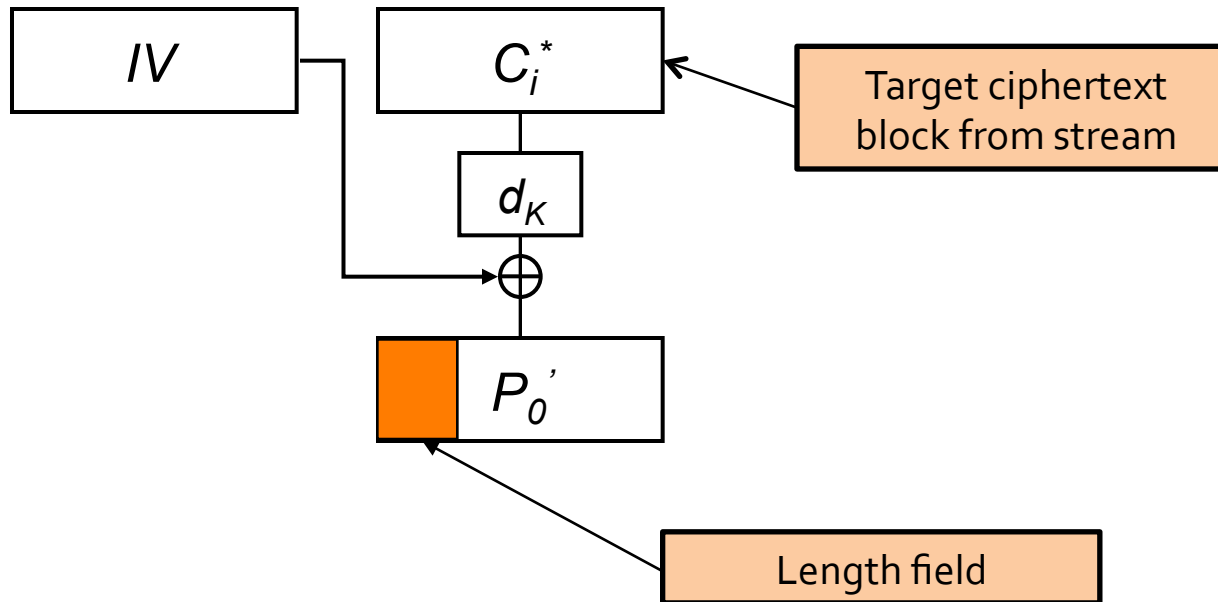


- How does decryption work?
- Recall: receiver gets a stream of bytes, and a single ciphertext can be fragmented over several TCP messages.

Breaking CBC mode in SSH [APWog]



Breaking CBC mode in SSH [APWog]

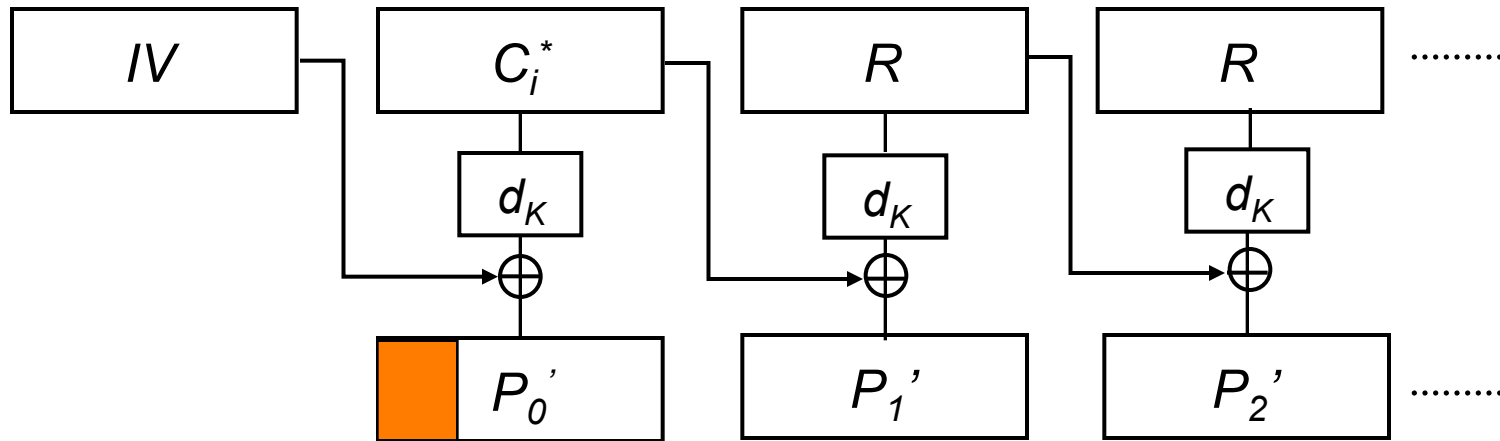


- The receiver will treat the first 32 bits of the calculated plaintext block as the packet length field for the new packet.
- Here:

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

where IV is known.

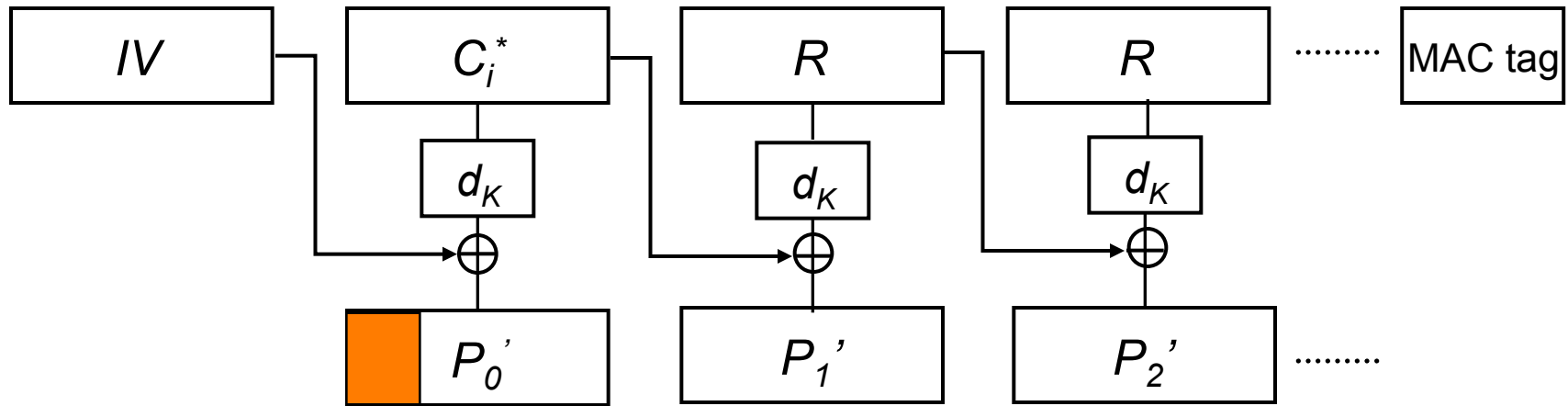
Breaking CBC mode in SSH [APWog]




The attacker then feeds random blocks to the receiver

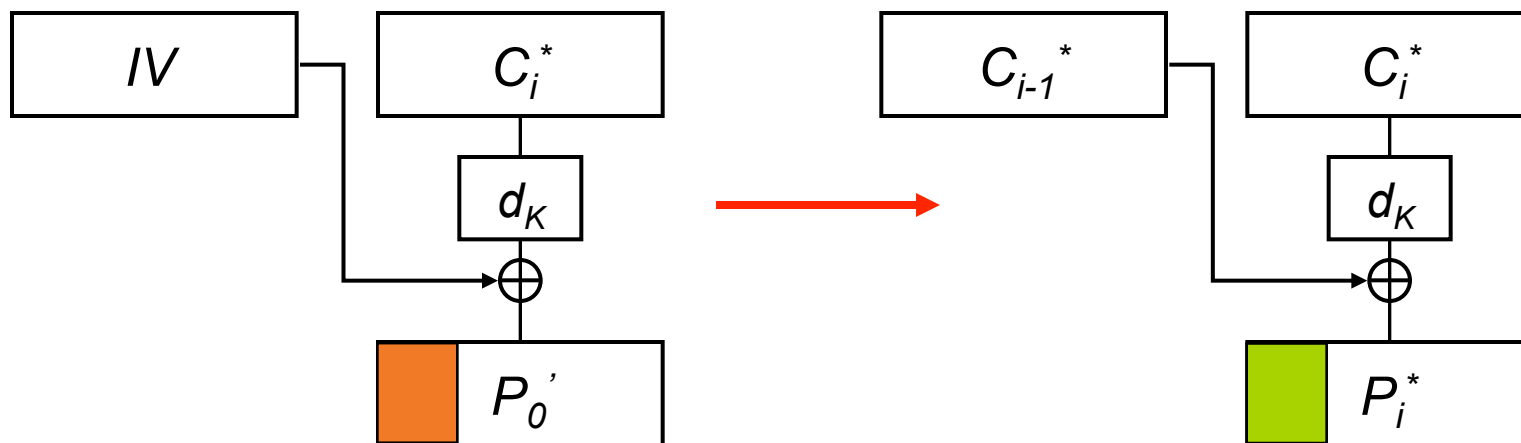
- One block at a time, waiting to see what happens at the server when each new block is processed
- This is possible because SSH runs over TCP and tries to do online processing of incoming blocks

Breaking CBC mode in SSH [APWog]



- 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 CBC mode in SSH [APWog]



- Knowing IV and 32 bits of P_0' , the attacker can now recover 32 bits of the target plaintext block P_i^* :

$$P_i^* = C_{i-1}^* \oplus d_K(C_i^*) = C_{i-1}^* \oplus IV \oplus P_0'$$

- Attack is slightly different in practice: implementation-specific length checks.

Security Modelling Implications?

- The attack works with random IVs too, invalidating the security proof in [BKNo2].
- The stateful AE notions used in [BKNo2] were for *atomic* ciphertext processing.
- But SSH permits *fragmented delivery* of ciphertexts.
- Oops!

Countermeasures to the attack

- **Abandon CBC-mode?**
 - Alternatives available at that time: CTR, RC₄.
 - Dropbear implemented CTR and relegated CBC mode in version 0.53.
- **Patch CBC-mode?**
 - Versions prior to OpenSSH5.1 affected.
 - OpenSSH5.2 also introduced a patch to stop the specific attack on CBC mode.
- **Develop new modes?**
 - Modes based on Generic EtM, AES-GCM, ChaCha20-Poly1305 were subsequently added to OpenSSH.
 - Mode proliferation!

AEAD in SSH today?

- In [ADHP16], we perform a measurement study of SSH deployment.
- We conducted two IPv4 address space scans in Nov/Dec 2015 and Jan 2016 using ZGrab/ZMap.
- Grabbing banners and SSH servers' preferred ciphers.
 - Actual cipher used in a given SSH connection depends on client and server preferences.
- Roughly 2^{24} servers found in each scan.
- Nmap fingerprinting suggests mostly embedded routers, firewalls.

The state of AEAD in SSH today: SSH versions

software	scan 2015-12		scan 2016-01	
dropbear_2014.66	7,229,491	(42.0%)	8,334,758	(47.0%)
OpenSSH_5.3	2,108,738	(12.3%)	2,133,772	(12.0%)
OpenSSH_6.6.1p1	1,198,987	(7.0%)	1,124,914	(6.3%)
OpenSSH_6.0p1	554,295	(3.2%)	573,634	(3.2%)
OpenSSH_5.9p1	467,899	(2.7%)	500,975	(2.8%)
dropbear_2014.63	422,764	(2.5%)	197,353	(1.1%)
dropbear_0.51	403,923	(2.3%)	434,839	(2.5%)
dropbear_2011.54	383,575	(2.2%)	64,666	(0.4%)
ROSSSH	345,916	(2.0%)	333,992	(1.9%)
OpenSSH_6.6.1	338,787	(2.0%)	252,856	(1.4%)
dropbear_0.46	301,913	(1.8%)	335,425	(1.9%)
OpenSSH_5.5p1	262,367	(1.5%)	272,990	(1.5%)
OpenSSH_6.7p1	261,867	(1.5%)	213,843	(1.2%)
OpenSSH_6.2	255,088	(1.5%)	288,710	(1.6%)
dropbear_2013.58	236,409	(1.4%)	249,284	(1.4%)
dropbear_0.53	217,970	(1.3%)	213,670	(1.2%)
dropbear_0.52	132,668	(0.8%)	136,196	(0.8%)
OpenSSH	110,602	(0.6%)	108,520	(0.6%)
OpenSSH_5.8	88,258	(0.5%)	89,144	(0.5%)
OpenSSH_5.1	86,338	(0.5%)	44,170	(0.2%)
OpenSSH_5.3p1	84,559	(0.5%)	0	(0.0%)
OpenSSH_7.1	83,793	(0.5%)	0	(0.0%)

The state of AEAD in SSH today: SSH versions

software	scan 2015-12		scan 2016-01	
dropbear_2014.66	7,229,491	(42.0%)	8,334,758	(47.0%)
OpenSSH_5.3	2,108,738	(12.8%)	2,133,000	(11.7%)
OpenSSH_6.6.1p1	1,198,987	(7.0%)	1,198,987	(6.6%)
OpenSSH_6.0p1	554,295	(3.2%)	554,295	(3.0%)
OpenSSH_5.9p1	467,899	(2.7%)	500,000	(2.7%)
dropbear_2014.63	422,764	(2.5%)	197,000	(1.1%)
dropbear_0.51	403,923	(2.3%)	434,000	(2.4%)
dropbear_2011.54	383,575	(2.2%)	64,000	(0.3%)
ROSSSH	345,916	(2.0%)	333,992	(1.9%)
OpenSSH_6.6.1	338,787	(2.0%)	252,856	(1.4%)
dropbear_0.46	301,913	(1.8%)	335,425	(1.9%)
OpenSSH_5.5p1	262,367	(1.5%)	272,990	(1.5%)
OpenSSH_6.7p1	261,867	(1.5%)	213,843	(1.2%)
OpenSSH_6.2	255,088	(1.5%)	288,710	(1.6%)
dropbear_2013.58	236,409	(1.4%)	249,284	(1.4%)
dropbear_0.53	217,970	(1.3%)	213,670	(1.2%)
dropbear_0.52	132,668	(0.8%)	136,196	(0.8%)
OpenSSH	110,602	(0.6%)	108,520	(0.6%)
OpenSSH_5.8	88,258	(0.5%)	89,144	(0.5%)
OpenSSH_5.1	86,338	(0.5%)	44,170	(0.2%)
OpenSSH_5.3p1	84,559	(0.5%)	0	(0.0%)
OpenSSH_7.1	83,793	(0.5%)	0	(0.0%)

Dropbear at 56-58%.
886,000 older than
version 0.53, so
vulnerable to variant of
2009 CBC-mode attack!

The state of AEAD in SSH today: SSH versions


software	scan 2015-12		scan 2016-01	
dropbear_2014.66	7,229,491	(42.0%)	8,334,758	(47.0%)
OpenSSH_5.3	2,108,738	(12.3%)	2,133,772	(12.0%)
OpenSSH_6.6.1p1	1,198,987	(7.0%)	1,124,914	(6.3%)
OpenSSH_6.0p1	554,295	(3.2%)	573,634	(3.2%)
OpenSSH_5.9p1	467,899	(2.7%)	500,077	(2.8%)
dropbear_2014.63	422,764	(2.4%)	190,000	(1.0%)
dropbear_0.51	403,923	(2.3%)	300,000	(1.6%)
dropbear_2011.54	383,575	(2.2%)	300,000	(1.6%)
ROSSSH	345,916	(2.0%)	300,000	(1.6%)
OpenSSH_6.6.1	338,787	(2.0%)	200,000	(1.1%)
dropbear_0.46	301,913	(1.8%)	300,000	(1.6%)
OpenSSH_5.5p1	262,367	(1.5%)	200,000	(1.1%)
OpenSSH_6.7p1	261,867	(1.5%)	200,000	(1.1%)
OpenSSH_6.2	255,088	(1.5%)	288,710	(1.6%)
dropbear_2013.58	236,409	(1.4%)	249,284	(1.4%)
dropbear_0.53	217,970	(1.3%)	213,670	(1.2%)
dropbear_0.52	132,668	(0.8%)	136,196	(0.8%)
OpenSSH	110,602	(0.6%)	108,520	(0.6%)
OpenSSH_5.8	88,258	(0.5%)	89,144	(0.5%)
OpenSSH_5.1	86,338	(0.5%)	44,170	(0.2%)
OpenSSH_5.3p1	84,559	(0.5%)	0	(0.0%)
OpenSSH_7.1	83,793	(0.5%)	0	(0.0%)

OpenSSH at 37-39%.
130,000-166,000 older
than version 5.2 and
prefer CBC mode, so
vulnerable to 2009
attack!

The OpenSSH patch

- OpenSSH patch, in version 5.2 and up:
 - If the length checks fail, do not send an error message, but wait until 2^{18} bytes have arrived, **then check the MAC**.
 - If the length checks pass, but **the MAC check eventually fails**, then wait until 2^{18} bytes have arrived, **then check the MAC**.
- **One** MAC check is done if length checks fail: on 2^{18} bytes.
- **Two** MAC checks are done if length checks pass: one on roughly LF bytes, the other on 2^{18} bytes.

Attacking the OpenSSH patch [ADHP16]

- This leads to a timing attack on CBC mode in OpenSSH5.2 and up, recovering up to 30 bits of plaintext from target block [ADHP16]. 
- Size of timing difference:
 - A MAC computation on roughly 2^{17} bytes (the expected value of LF).
 - About 2000 times bigger than the Lucky 13 timing difference!
- Affects roughly 20,000 OpenSSH servers.

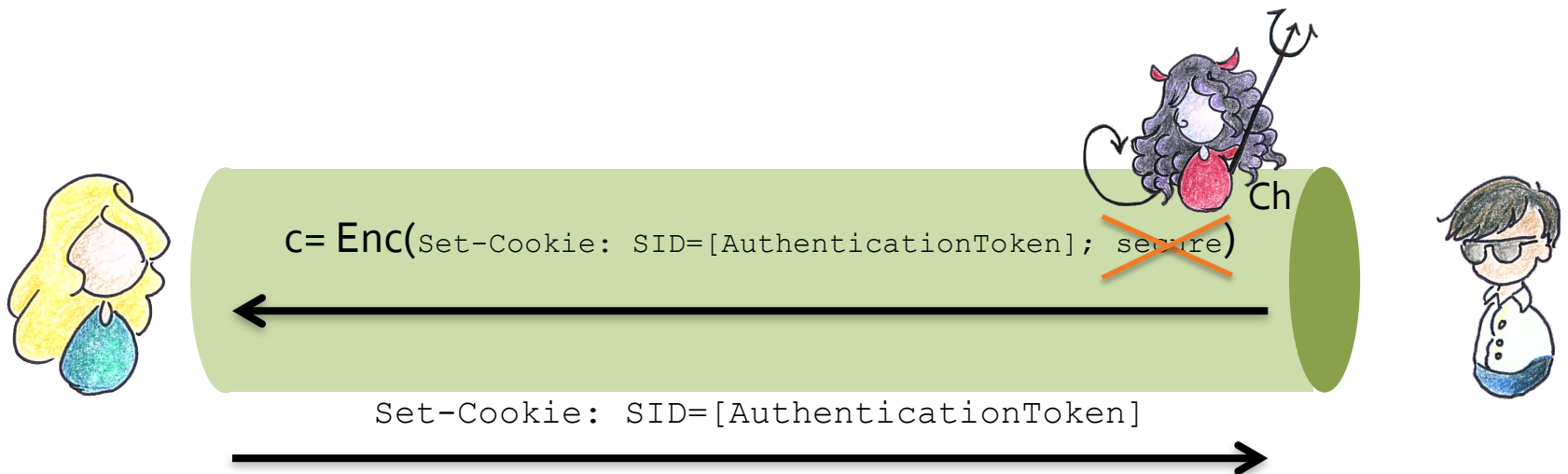
Disclosure of the attack

- We notified the OpenSSH team of the attack on 5th May 2016.
- They are considering adding countermeasures for the next release of OpenSSH (7.3).
- “...we do not feel that an emergency release is necessary, nor that the attack remain secret ahead of such a release.”
- OpenSSH has steadily been deprecating old algorithms and modes.
- CBC mode was already disabled by default in OpenSSH 6.7 (but can be re-enabled).
- But OpenSSH cannot force people to stop using old versions of the software.
 - The legacy problem – not unique to SSH.

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?

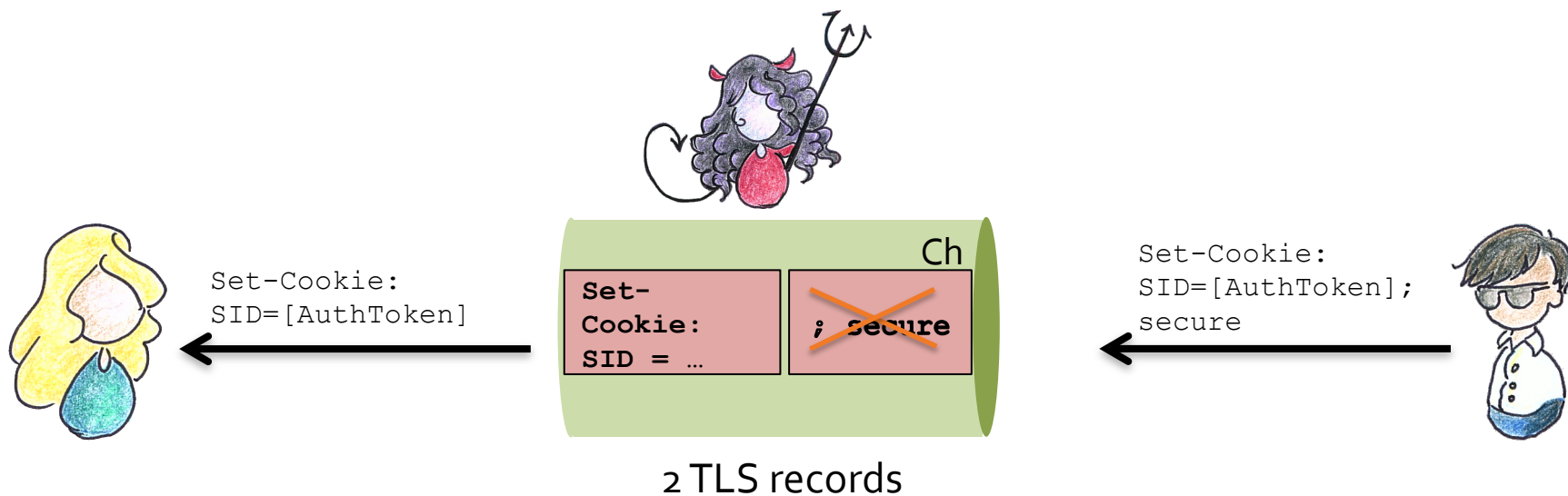
6.2.1. Fragmentation

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)

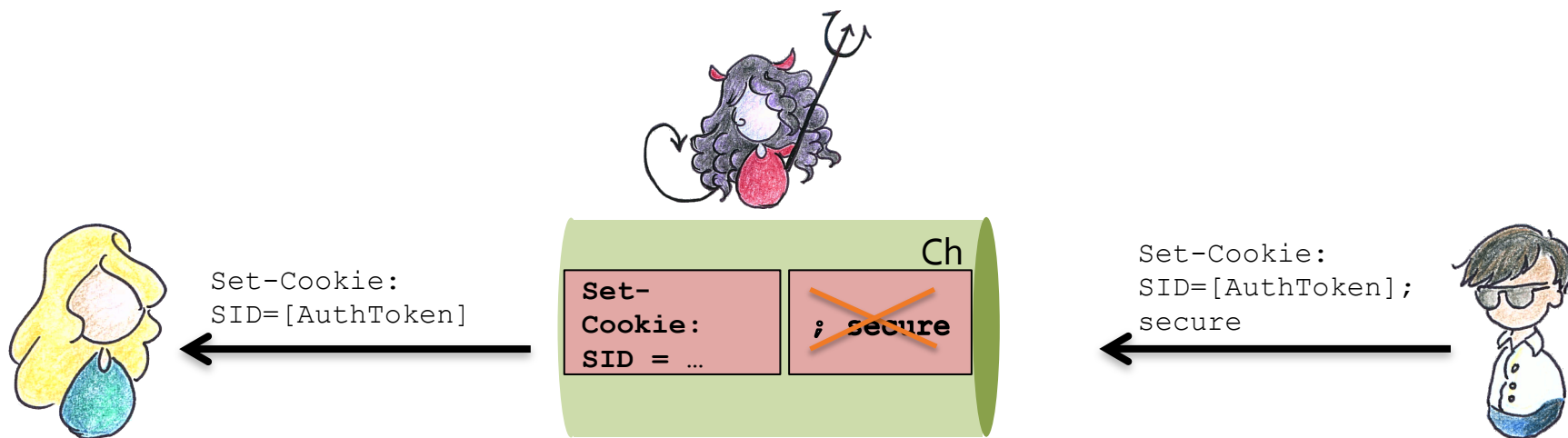
Cookie cutters

- So SSL/TLS can (and will) fragment when *sending*.
- Compare to SSH that has to deal with fragments only 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 –it seems that developers do not understand the interface provided by SSL/TLS?





Building Better Models

Motivation: AEAD in OpenSSH today

encryption and mac algorithm		count
aes128-ctr + hmac-md5	3,877,790	(57.65%)
aes128-ctr + hmac-md5-etm@	2,010,936	(29.90%)
aes128-ctr + umac-64-etm@	331,014	(4.92%)
aes128-cbc + hmac-md5	161,624	(2.40%)
chacha20-poly1305@	115,526	(1.72%)
aes128-ctr + hmac-sha1	68,027	(1.01%)
des + hmac-md5	40,418	(0.60%)
aes256-gcm@	28,019	(0.42%)
aes256-ctr + hmac-sha2-512	17,897	(0.27%)
aes128-cbc + hmac-sha1	11,082	(0.16%)
aes128-ctr + hmac-ripemd160	10,621	(0.16%)

OpenSSH preferred algorithms

- Lots of diversity, surprising amount of “generic EtM” (gEtM).
- CTR dominates, followed by CBC.
- ChaCha20-Poly1305 on the rise? (became default in OpenSSH 6.9).
- Small amount of GCM.

Analysis of SSH-CTR

- [PW10] developed a bespoke security model for CTR mode in SSH and proved it secure (assuming block cipher is a PRP).
- The model allows the attacker to deliver ciphertexts to decryption oracle in a byte-by-byte fashion.
- Accurately models OpenSSH's CTR mode implementation.
 - Sanity checking of length field, with related error messages, MAC failures, etc.
 - Complex pseudo-code descriptions of algorithms and oracles.

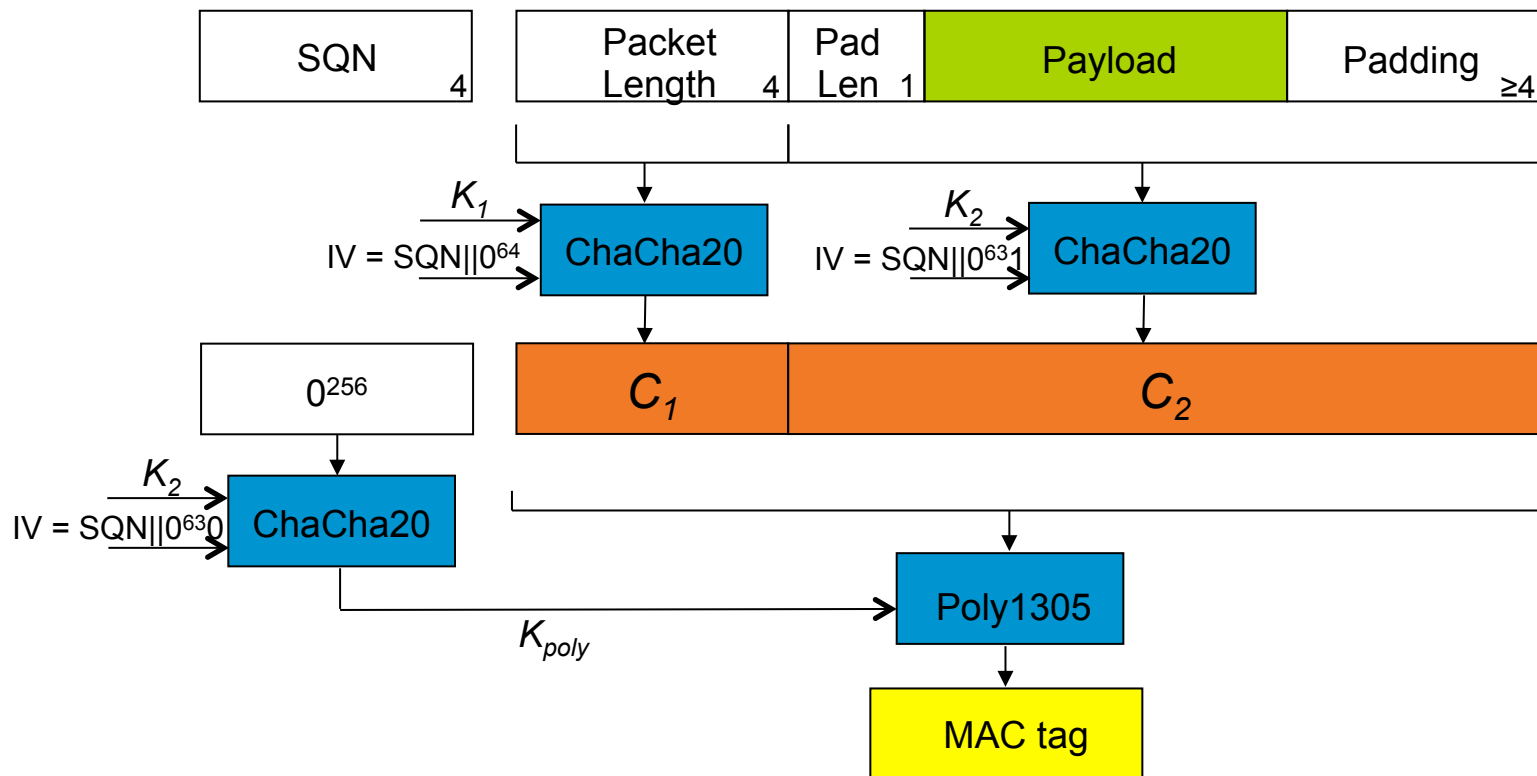
Symmetric Encryption Supporting Fragmented Decryption

- [BDPS₁₂] developed a general framework for studying “Symmetric Encryption schemes supporting fragmented decryption” like SSH.
- Their IND-CFA model allows the attacker to deliver ciphertext to a decryption oracle in a symbol-by-symbol fashion and observe any errors/message outputs.
- [BDPS₁₂] also identified additional security properties that SSH *attempts* to provide:
 - Boundary Hiding (BH) and Denial-of-Service resistance.

Developing and Using the Models

- [FGMP15] developed a framework for studying **Streaming Secure Channels** like TLS, which permit fragmentation both in sending and receiving.
 - cf. work on TLS mentioned by Cedric Fournet this morning.
 - Cryptographic-game-based rather than type-based.
- [ADHP16] uses the framework of [BDPS12] to study gEtM, AES-GCM, and ChaCha20-Poly1305 in OpenSSH.
 - Identifies a bug in the [BDPS12] security model.
 - Proves security of all modes.
 - Finds an error in gEtM: MAC computed before decryption but not checked until after decryption!

ChaCha20-Poly1305 in OpenSSH





Closing remarks

Closing remarks

- Simple security models for symmetric encryption *versus* complex security properties desired of secure channels.
- There is still a rich research seam to mine here.

"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning."

Closing remarks

