Bar-Ilan Winter School
Lecture 5
Attacks and security notions for the SSH secure channel

Kenny Paterson @kennyog
Based on joint work with Martin Albrecht, Jean Paul Degabriele and Torben Hansen

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Overview

1. Introducing SSH
2. SSH measurement study
3. An unfortunate sequence of attacks on CBC-mode in OpenSSH
4. Security models for the SSH secure channel
5. Security analysis of other SSH and OpenSSH modes – CTR, ChaChaPoly, gEtM, AES-GCM
6. Better security for SSH: InterMAC

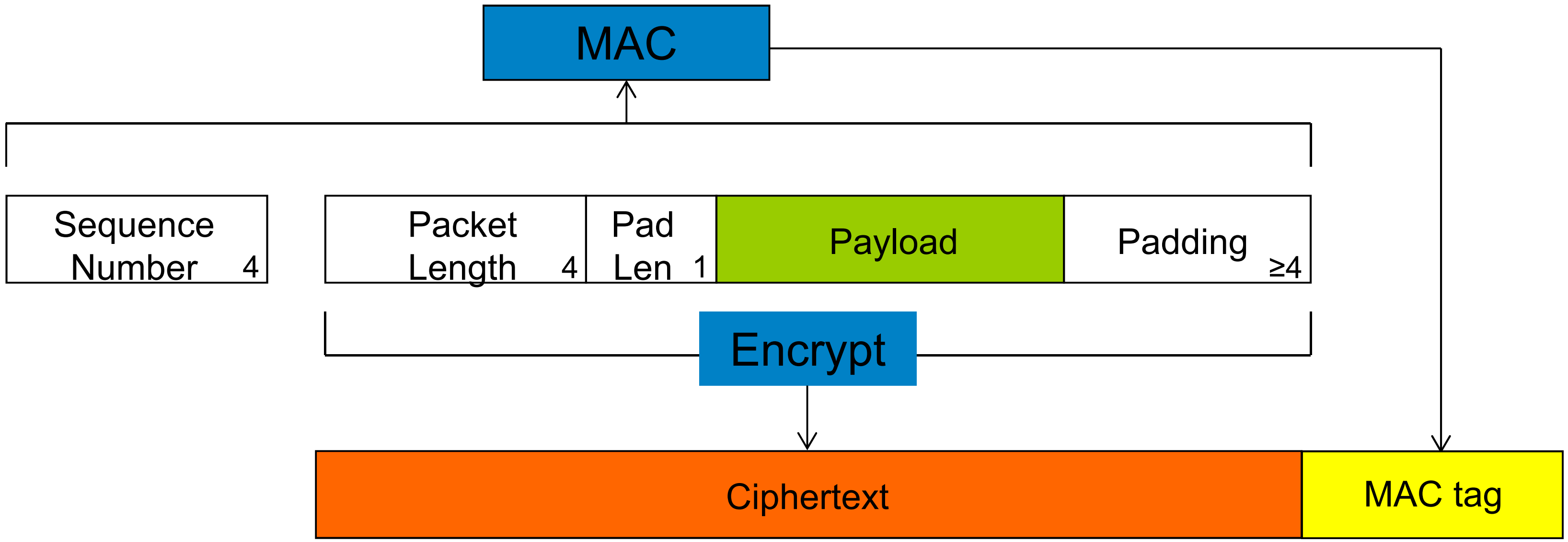
Introducing SSH and related work

Introduction to SSH

Secure Shell or SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices. Used primarily on Linux and Unix based systems to access shell accounts, SSH was designed as a replacement for TELNET and other insecure remote shells, which send information, notably passwords, in plaintext, leaving them open for interception. ***The encryption used by SSH provides confidentiality and integrity of data over an insecure network, such as the Internet****.*

– Wikipedia

SSH Binary Packet Protocol



- Stateful Encode-then-E&M construction
- Packet length field measures the size of the packet: |PadLen|+ |Payload| + |Padding|.
- RFC 4253 (2006): various block ciphers in **CBC mode (with chained IV)** and **RC4**.
- RFC 4344 (2006): added **CTR mode** for the corresponding block ciphers.

Timeline of related work on SSH BPP

2002.

- Formal security analysis of SSH BPP by Bellare, Kohno and Namprempre [BKN02]: introduce **stateful security notions for symmetric encryption** and proved **SSH-CTR** and **SSH-CBC variants** (w/o IV chaining) secure.

2009.

- Albrecht, Paterson and Watson [APW09] discover a plaintext-recovery attack against **SSH in CBC mode**.
- The attack exploits **fragmented delivery in TCP/IP**, and works on **all CBC variants** considered in [BKN02].
- The then leading implementation was OpenSSH (reported 80% of servers); OpenSSH team release a **patch** in version 5.2 to stop the specific attack.

Timeline of related work on SSH BPP

2010.

- The [APW09] attack highlights deficiencies in the [BKN02] security model.
- Paterson and Watson [PW10] prove SSH-CTR secure in an extended security model that allows adversary to deliver fragmented ciphertexts.

2012.

- Boldyreva, Degabriele, Paterson and Stam [BDPS12] study ciphertext fragmentation more generally, addressing limitations in the [PW10] model, introducing **IND-CFA security**.
- [BDPS12] also considers **boundary hiding** and resistance to a special type of **denial of service** attack as additional security requirements.

SSH measurement study

SSH measurement study

- In [ADHP16], we performed a measurement study of SSH deployment.
- We conducted two complete IPv4 address space scans in Nov/Dec 2015 and Jan 2016 using ZGrab/Zmap.
  - Grabbing banners and SSH servers' preferred algorithms.
  - 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 and firewall devices.
- Data available at:

  https://bitbucket.org/malb/a-surfeit-of-ssh-cipher-suites/overview

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%)	[illegible],144	[illegible]
OpenSSH_5.1	86,338	(0.5%)	44,[illegible]	[illegible]
OpenSSH_5.3p1	84,559	(0.5%)	0	[illegible]
OpenSSH_7.1	83,793	(0.5%)	0	[illegible]

Mostly OpenSSH and dropbear; others less than 5%.

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%)	[illegible]	[illegible]
ROSSSH	345,916	(2.0%)	[illegible]	[illegible]
OpenSSH_6.6.1	338,787	(2.0%)	[illegible]	[illegible]
dropbear_0.46	301,913	(1.8%)	335,425	[illegible]
OpenSSH_5.5p1	262,367	(1.5%)	272,990	[illegible]
OpenSSH_6.7p1	261,867	(1.5%)	213,843	[illegible]
OpenSSH_6.2	255,088	(1.5%)	288,710	[illegible]
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%. 886k older than version 0.52, so vulnerable to variant of 2009 CBC-mode attack.

The state of 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%)	[illegible]	[illegible]
dropbear_0.53	217,970	(1.3%)	[illegible]	[illegible]
dropbear_0.52	132,668	(0.8%)	[illegible]	[illegible]
OpenSSH	110,602	(0.6%)	[illegible]	[illegible]
OpenSSH_5.8	88,258	(0.5%)	[illegible]	[illegible]
OpenSSH_5.1	86,338	[illegible]	[illegible]	[illegible]
OpenSSH_5.3p1	84,559	(0.5%)	[illegible]	[illegible]
OpenSSH_7.1	83,793	(0.5%)	[illegible]	[illegible]

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

SSH versions

- Dropbear dominates over OpenSSH.
- Long tail of old software versions.
    - Most popular version of OpenSSH was version 5.3, released Oct 2009 (current version is 7.5).
    - Determined by major Linux distros?
- Non-negligible percentage of Dropbear and OpenSSH servers were potentially still vulnerable to the 2009 attack.
    - 8.4% for Dropbear.

OpenSSH preferred algorithms

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 ("@" = "@openssh.com")

- Lots of diversity (155 different combinations).
- CTR dominates, followed by CBC, surprising amount of EtM.
- ChaCha20-Poly1305 on the rise? (became default in OpenSSH 6.9).
- Small amount of GCM.

Dropbear preferred algorithms

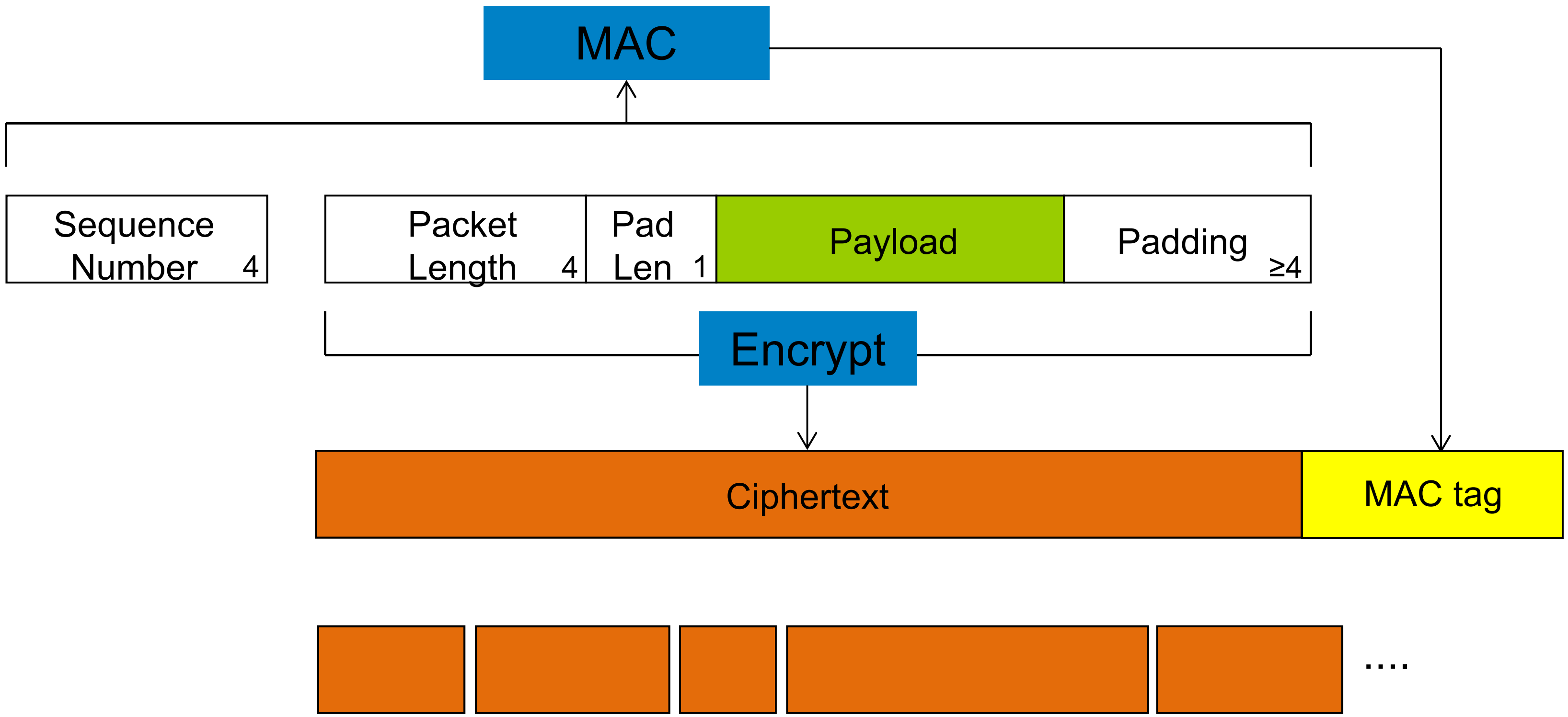
encryption and mac algorithm	count	
aes128-ctr + hmac-sha1-96	8,724,863	(90.44%)
aes128-cbc + hmac-sha1-96	478,181	(4.96%)
3des-cbc + hmac-sha1	321,492	(3.33%)
aes128-ctr + hmac-sha1	62,465	(0.65%)
aes128-ctr + hmac-sha2-256	36,150	(0.37%)
aes128-cbc + hmac-sha1	14,477	(0.15%)

Dropbear preferred algorithms

- Less diversity than OpenSSH.
- CTR also dominates, followed by CBC.
- No "exotic" options.
- All CBC modes unpatched against variant of 2009 attack (8.4%).

An unfortunate sequence of attacks on CBC mode in OpenSSH

SSH Binary Packet Protocol

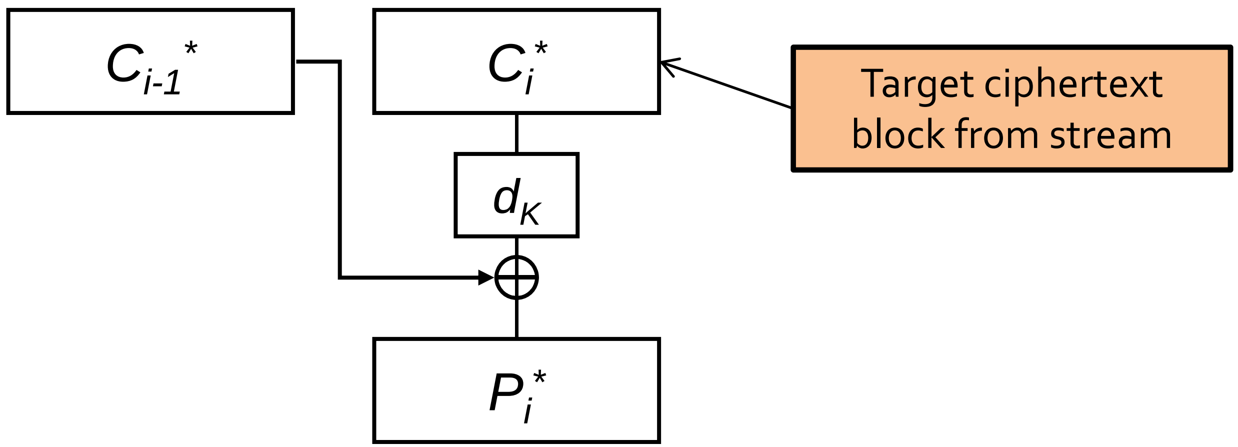


How would you perform decryption for an incoming sequence of ciphertext fragments?

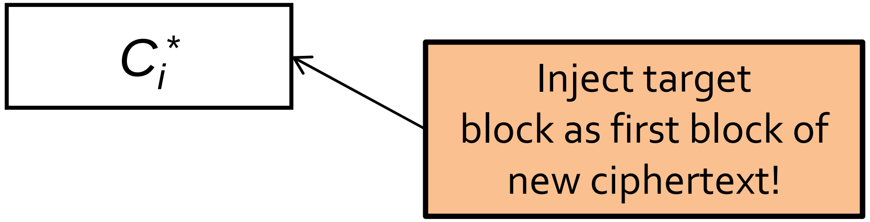
The [APW09] attack (simplified)

- Decryption in OpenSSH CBC mode (prior to 5.2):
  - Use a buffer to hold the incoming sequence of ciphertext fragments.
  - Decrypt the fragments block-by-block as they arrive.
  - 4-byte packet length field LF is obtained from the **first** block of the **first** fragment to be received.
  - Continue to buffer+decrypt until a total of LF+|MAC| bytes have been received.
  - Verify the MAC on SQN || PTXT (with connection termination and error message if MAC verification fails).

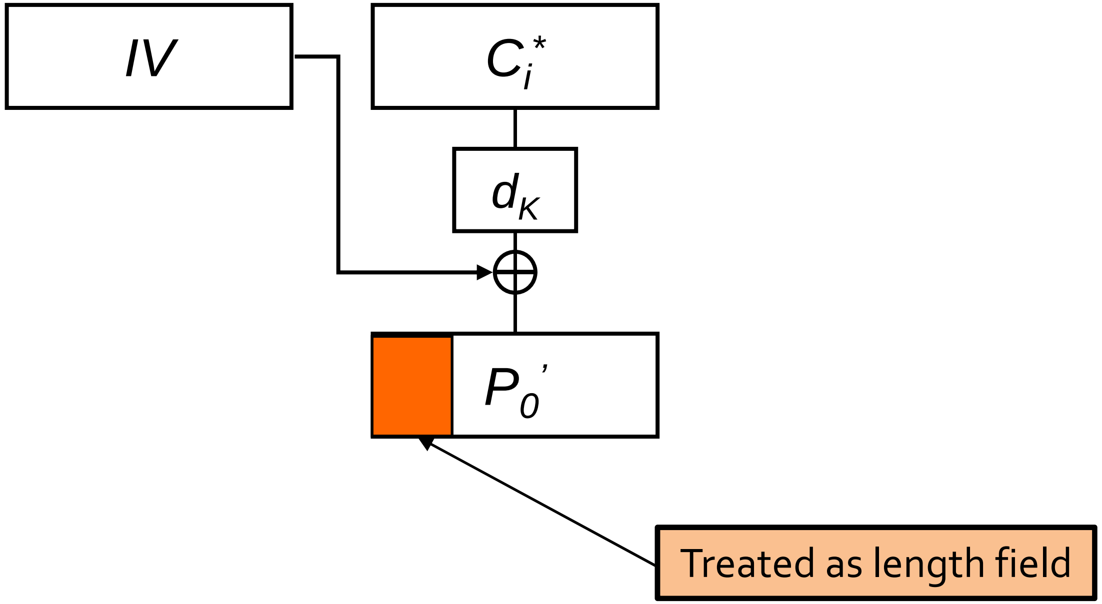
Breaking CBC mode in SSH [APW09]



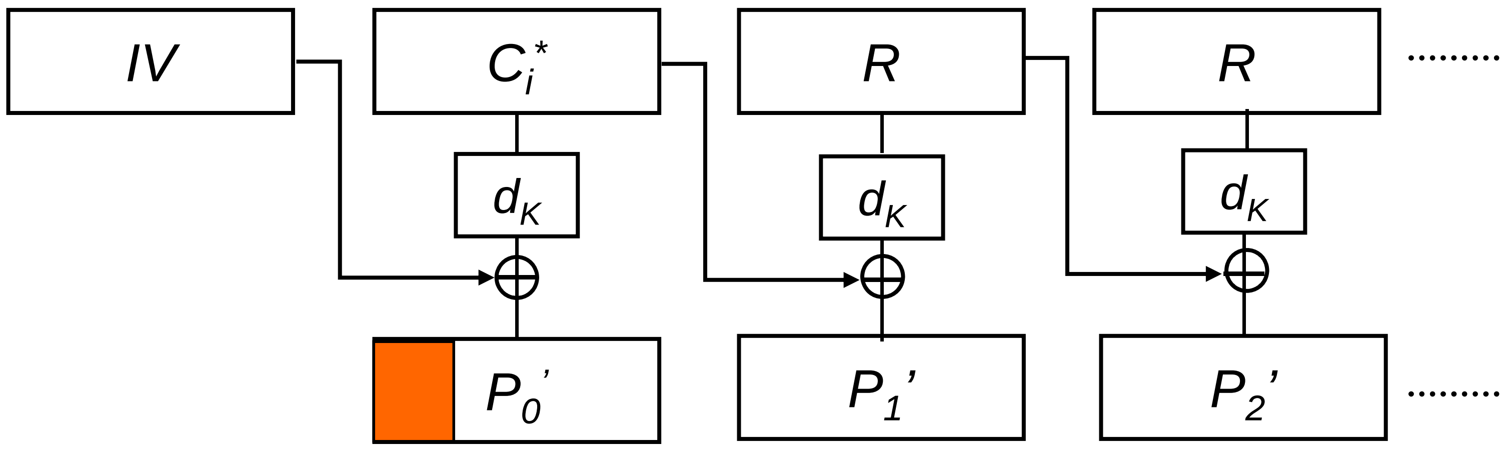
Breaking CBC mode in SSH [APW09]



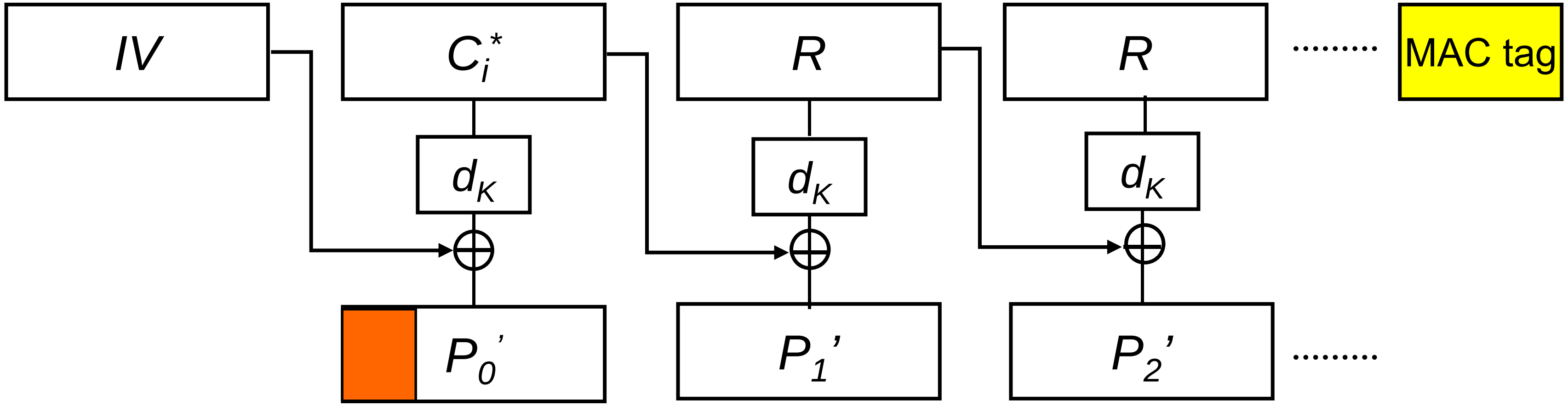
Breaking CBC mode in SSH [APW09]

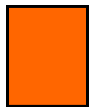


Breaking CBC mode in SSH [APW09]

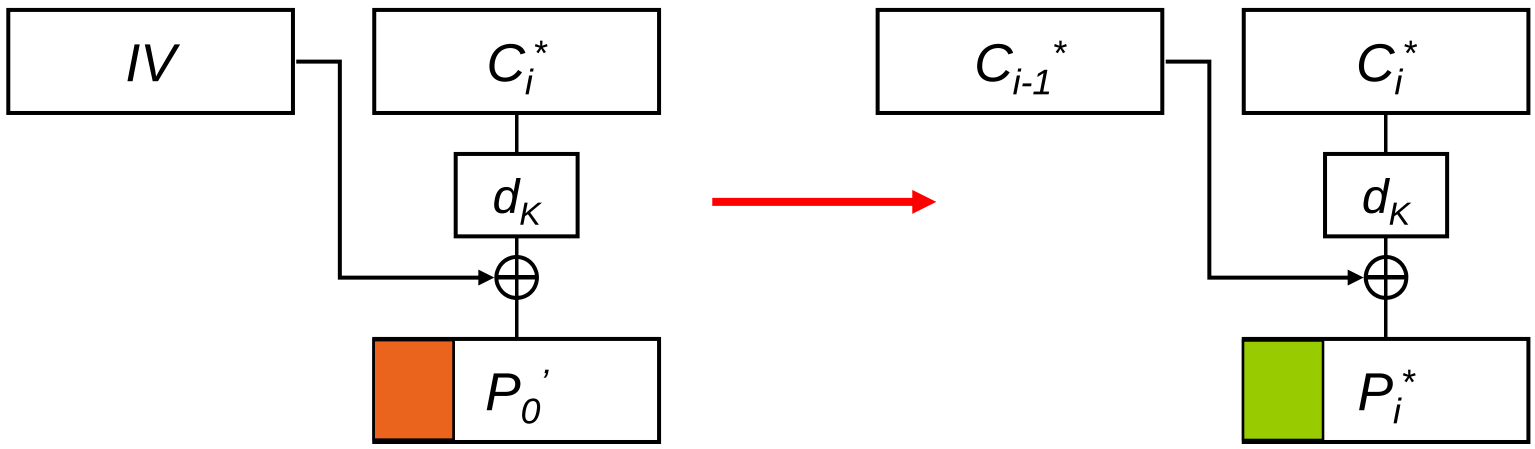


Breaking CBC mode in SSH [APW09]



- Once **enough** data has arrived, the receiver will get what it thinks is the MAC tag
  - The MAC verification will fail with overwhelming probability
  - So the connection is terminated (with an error message)
- **Question**: 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 [APW09]



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

$$\text{LF} \oplus [IV]_{0..3} = \square \oplus [C_{i-1}^{*}]_{0..3}$$

The [APW09] attack (less simplified)

- OpenSSH5.1 actually performs two sanity checks on the length field when decrypting the first ciphertext block:
  - Check 1: $5 \leq LF \leq 2^{18}$.
  - Check 2: total length ($LF+4$) is a multiple of the block size:

$$LF + 4 \bmod BL = 0.$$

- Each check produces a *different* error message on the network, distinguishable by attacker.
- If both checks pass, then OpenSSH waits for more bytes, then performs MAC check, resulting in a third distinct error message.
- The different error messages allow up to 32 bits of plaintext to be recovered with probability 2^{-18}.

OpenSSH 5.2 patch against [APW09] attack

Sanity checks:

$5 \leq LF \leq 2^{18}$

$LF + 4 \bmod BL = 0$

→ FAIL → ~~*ssh2_msg_disconnect*~~

→ Wait until 2^{18} bytes have arrived, then check a MAC on 2^{18} bytes.

↓

PASS

↓

Wait for LF+|MAC| bytes

↓

VERIFY → FAIL → ~~*"corrupted MAC on input"*~~

→ Wait until 2^{18} bytes have arrived, then check a MAC on 2^{18} bytes.

No error message is sent until 2^{18} bytes of ciphertext have arrived.

Is this a good patch?

OpenSSH 5.2 patch against [APW09] attack

Sanity checks:

$5 \leq LF \leq 2^{18}$ → FAIL → ~~*ssh2_msg_disconnect*~~

$LF + 4 \bmod BL = 0$

→ Wait until 2^{18} bytes have arrived, then check a MAC on 2^{18} bytes.

↓

PASS

↓

Wait for LF+|MAC| bytes

MAC on 2^{18} bytes

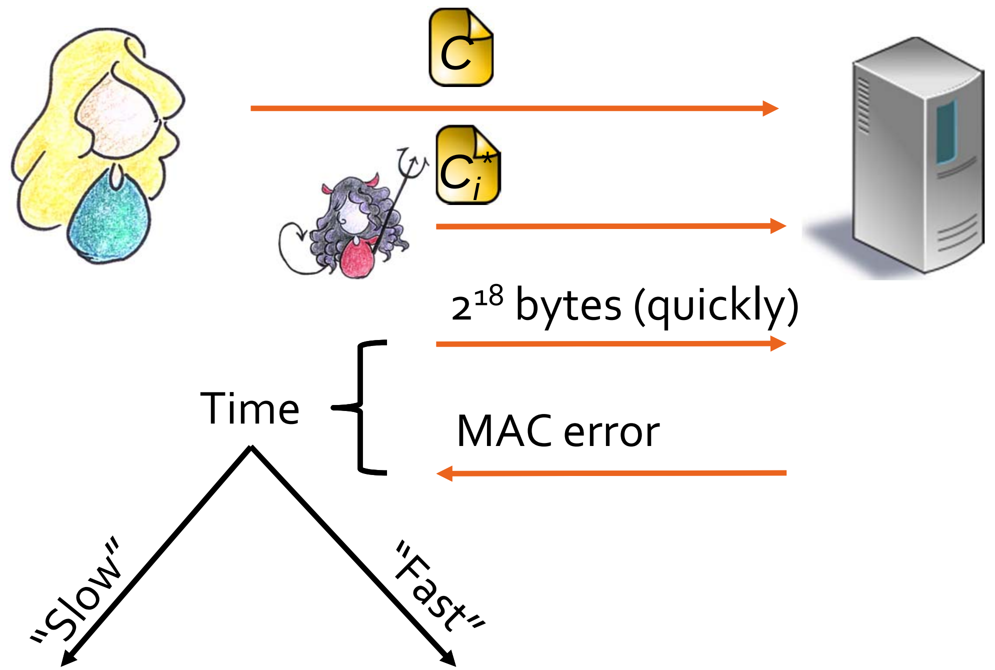
↓

VERIFY → FAIL → ~~*"corrupted MAC on input"*~~

MAC on ~LF bytes + MAC on 2^{18} bytes → Wait until 2^{18} bytes have arrived, then check a MAC on 2^{18} bytes.

No error message is sent until 2^{18} bytes of ciphertext have arrived.

[ADHP16] attack against the OpenSSH 5.2 patch



MAC on ~LF bytes +
MAC on 2^{18} bytes
Sanity check PASS

MAC on 2^{18} bytes
Sanity check FAIL

- Attacker can distinguish PASS/FAIL conditions, leaking 18 bits of plaintext.
- With careful timing, attacker can recover ~30 bits of plaintext.

OpenSSH 7.3 patch against [ADHP16] attack

Sanity checks: $5 \leq LF \leq 2^{18}$, $LF + 4 \bmod BL = 0$ → FAIL → ~~*ssh2_msg_disconnect*~~

→ Wait until 2^{18} bytes have arrived, then check a MAC on 2^{18} bytes.

MAC on 2^{18} bytes

↓

PASS

↓

Wait for LF+|MAC| bytes

↓

VERIFY → FAIL → ~~*"corrupted MAC on input"*~~

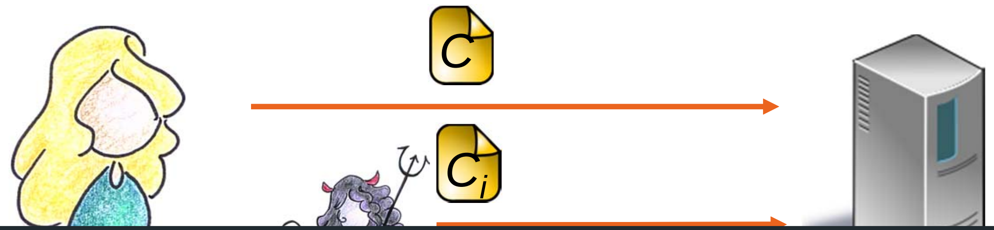
MAC on ~LF bytes +
MAC on 2^{18} - LF bytes

→ ~~Wait until 2^{18} bytes have arrived, then check a MAC on 2^{18} bytes.~~

→ Wait until 2^{18} bytes have arrived, then check a MAC on 2^{18} - LF bytes.

So is this a good patch?

Attacking the OpenSSH 7.3 patched patch



Our recommended patch actually made things significantly worse!

Performed

"Slow"

"Fast"

MAC on ~LF bytes +
MAC on 2^18 - LF bytes
Sanity check PASS

MAC on 2^18 bytes
Sanity check FAIL

Timing difference


I wonder if anyone noticed?
I'm not so sure!
I think we got away with it!
BEST PAPER AWARD
BEST PAPER AWARD

Disclosure of the attacks

- We first notified the OpenSSH team of the attack on the patch for the [APW09] attack on 5/5/2016.
- They first set of countermeasures in OpenSSH 7.3 (released 1/8/2016).
- We then notified OpenSSH of the new attack on 15/12/2016, along with some other, more subtle byte counting issues.
- These were partly addressed in OpenSSH 7.5 (released 20/3/2017).
- But several residual issues remain unpatched, **including the final attack**.
- In defence of OpenSSH:
  - OpenSSH has steadily been deprecating old algorithms and modes.
  - For example, CBC mode was already disabled by default in OpenSSH 6.7.

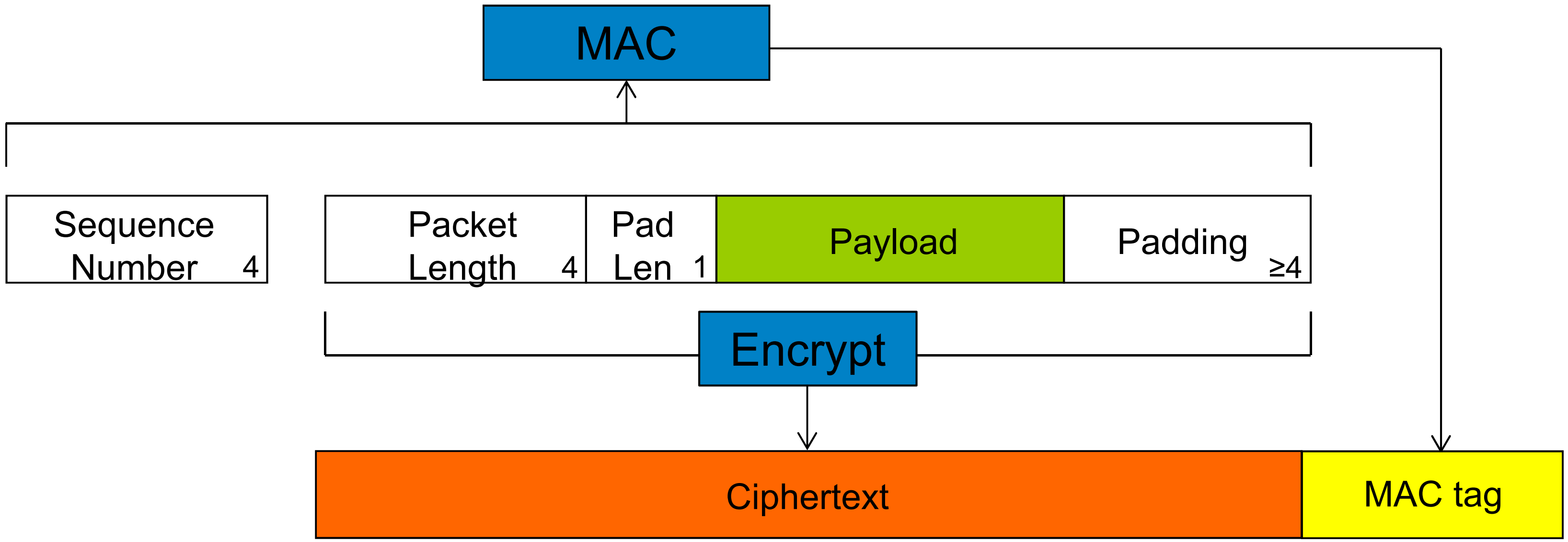
Security analysis of other SSH and OpenSSH modes – CTR, gEtM, AES-GCM, ChaCha20Poly1305

OpenSSH encryption modes

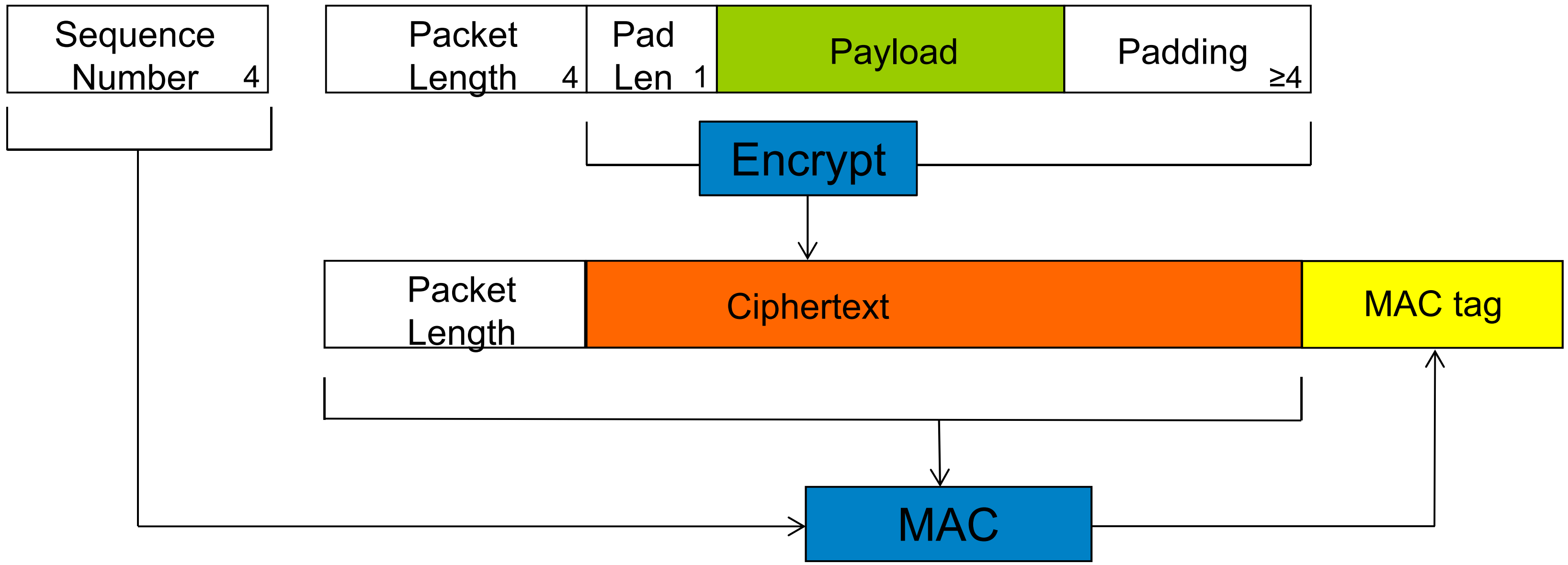
A number of new schemes have been introduced in OpenSSH since [APW09]:

- AES-GCM: since v6.2; **length field not encrypted** but is instead treated as associated data.
- generic Encrypt-then-MAC (gEtM): since v6.2; overrides native E&M processing; **length field not encrypted** but protected by MAC.
- ChaCha20-Poly1305@openssh.com: since v6.5 and promoted to default in v6.9; **reintroduces encryption of length field**.

Binary Packet Protocol native E&M construction



Binary Packet Protocol generic EtM construction



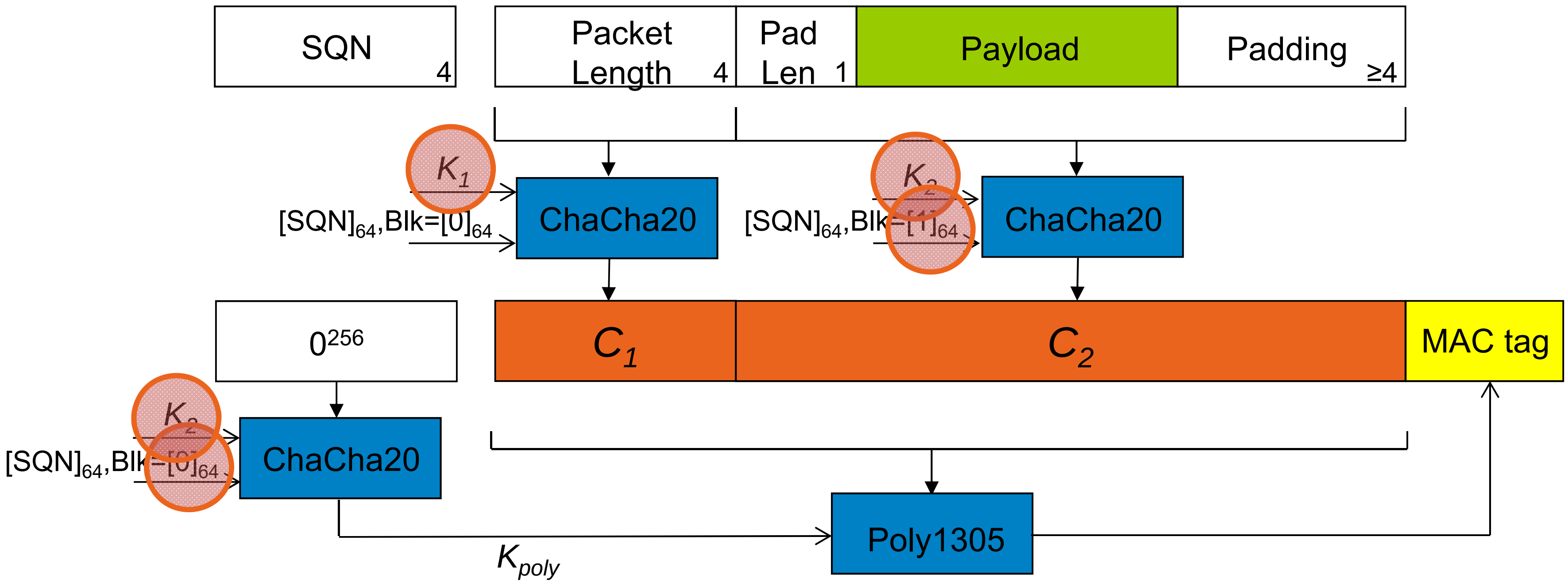
- Stateful Encode-then-EtM construction.
- AES-GCM works similarly.
- Note packet length field in the clear: construction gives up on hiding packet lengths.
- Code = documentation.

Binary Packet Protocol generic EtM security issue

```
/* EtM: compute mac over encrypted input */
if (mac && mac->enabled && mac->etm) {
    if ((r = mac_compute(mac, state->p_read.seqnr, ...
}
if ((r = sshbuf_reserve(state->incoming_packet, aadlen + need, ...
if ((r = cipher_crypt(&state->receive_context, state->p_read.seqnr, cp, ...
if ((r = sshbuf_consume(state->input, aadlen + need + authlen)) != 0)
    goto out;
/*
 * compute MAC over seqnr and packet,
 * increment sequence number for incoming packet
 */
if (mac && mac->enabled) {
    if (!mac->etm) ...
    if (timingsafe_bcmp(macbuf, sshbuf_ptr(state->input), ...
    }
```

- Sequence: compute MAC, then decrypt, then check MAC.
- Issue arises because of retrofitting gEtM in legacy E&M code.
- No concrete attack, but dangerous to decrypt unauthenticated ciphertext (cf. padding oracle attacks).
- Addressed in OpenSSH 7.3.

ChaCha20-Poly1305@openssh.com



- ChaCha20-Poly1305@openssh.com: since OpenSSH 6.5 and promoted to default in v6.9; **reintroduces encryption of length field**.
- OpenSSH developers seem to care a lot about **hiding packet lengths!**

Security analysis from [ADHP16]

- We used the **framework of [BDPS12]** for **symmetric encryption schemes supporting ciphertext fragmentation** to analyse the security of these schemes.
- We identified and fixed a **technical issue** in the IND-sfCFA confidentiality definition from [BDPS12].
- We introduced a matching notion of **ciphertext integrity**, INT-sfCTXT, which was not considered in [BDPS12].

Symmetric Schemes supporting ciphertext fragmentation: A flavour of the formal definitions

- [BDPS12] introduced a class of symmetric encryption (SE) schemes supporting ciphertext fragmentation.
- **KGen:** selects key K and sets initial encryption and decryption states σ_0, τ_0.
- **Enc:** takes complete plaintext and state as input and produces corresponding ciphertext and an updated state:

$$(c,\sigma') \leftarrow \text{Enc}(K,m,\sigma)$$

- **Dec:** takes arbitrary bit-strings (and state) as input, and produces bit-strings from $(\{0,1\}^* \cup \{\mathbf{P}\} \cup S_{err})^* \times \Sigma$
  - S_{err} : set of possible error symbols arising during decryption.
  - **P**: a distinguished "end of message" symbol.
  - Σ: state space of decryption algorithm.

Correctness for SE schemes supporting fragmentation

- Informally: “Decryption works properly across fragmented and concatenated ciphertexts”.
- Formally, for any sequence of calls to Enc:

$$(c_i, \sigma_i) \leftarrow \mathrm{Enc}(K, m_i, \sigma_{i-1}) \quad (\text{for } i=1,\ldots,t)$$

and any sequence of ciphertext fragments:

$$f_1, f_2, \ldots, f_n,$$

if $c_1 \| c_2 \| \ldots \| c_t$ is a prefix of $f_1 \| f_2 \| \ldots \| f_n$, and

$$(m'_i, \tau_i) \leftarrow \mathrm{Dec}(K, f_i, \tau_{i-1}) \quad (\text{for } i=1,\ldots,n)$$

then

$$m_1 \mathbf{P} m_2 \mathbf{P} \ldots m_3 \mathbf{P} \text{ is a prefix of } m'_1 \| m'_2 \| \ldots \| m'_n.$$

(NB other subtly different correctness definitions are possible!)

Security for SE schemes supporting ciphertext fragmentation

- Confidentiality and integrity notions extend those of [BKN02] for stateful setting.
- INDsf-CFA: indistinguishability of encryptions under a stateful, chosen fragment attack.
- Adversary has a regular encryption oracle, called on equal-length message pairs (m_0,m_1).
- Adversary has a decryption oracle accepting a sequence of fragments $f_1, f_2, \ldots$ as input.
- Decryption oracle suppresses output until sequence ‘goes out of sync’ with output of encryption oracle.

Security for SE schemes supporting ciphertext fragmentation [ADHP16]

alg. INI

sync $\leftarrow$ true
$i \leftarrow 0, j \leftarrow 0$
$C \leftarrow [\,], M \leftarrow [\,]$
$F \leftarrow \varepsilon, M' \leftarrow \varepsilon$
$b \twoheadleftarrow \{0, 1\}$
$(K, \sigma, \varrho) \leftarrow \mathcal{K}$
return

alg. LR(b, m_0, m_1)

if $|m_0| \neq |m_1|$ **return** ε
$(c, \sigma) \leftarrow \mathcal{E}_K(m_b, \sigma)$
$i \leftarrow i + 1, C[i] \leftarrow c$
$M[i] \leftarrow m_b \parallel \P$
return c

alg. ENC(m)

$(c, \sigma) \leftarrow \mathcal{E}_K(m, \sigma)$
$i \leftarrow i + 1, C[i] \leftarrow c$
$M[i] \leftarrow m \parallel \P$
return c

alg. DEC(f)

$(m, \varrho) \leftarrow \mathcal{D}_K(f, \varrho)$
$F \leftarrow F \parallel f, M' \leftarrow M' \parallel m$
if sync $=$ true
  $j \leftarrow \min(\{n \mid C[1 \ldots n] \not\preceq F\} \cup \{i\})$
  if $F \preceq C[1 \ldots j]$
    $m \leftarrow \varepsilon$
  else
    $m \leftarrow M' \,\%\, M[1 \ldots j-1]$
    if $C[1 \ldots j] \preceq F$
      $m \leftarrow M' \,\%\, M[1 \ldots j]$
    if $m \neq \varepsilon$
      sync $\leftarrow$ false
return m

Security analysis from [ADHP16]

	IND-sfCFA	INT-sfCTF	BH-CPA	BH-sfCFA	n-DOS-sfCFA
CBC	✗	✓	✓	✗	✗
fixed-CBC	✗	✓	✓	✗	✗
CTR	✓	✓	✓	✗	✗
fgEtM	✓	✓	✗	✗	✗
AES-GCM	✓	✓	✗	✗	✗
ChaCha20-Poly1305	✓	✓	✓	✗	✗

Security comparison of SSH AE modes

Additional goals from [BDPS12]:

- BH-CPA (passive adversary) – boundary hiding for passive attackers.
- BH-sfCFA (active adversary) – boundary hiding for active attackers.
- *n*-DOS-sfCFA: decryption must produce some output (plaintext or error) after receiving at most an *n*-bit sequence of fragments chosen by adversary.

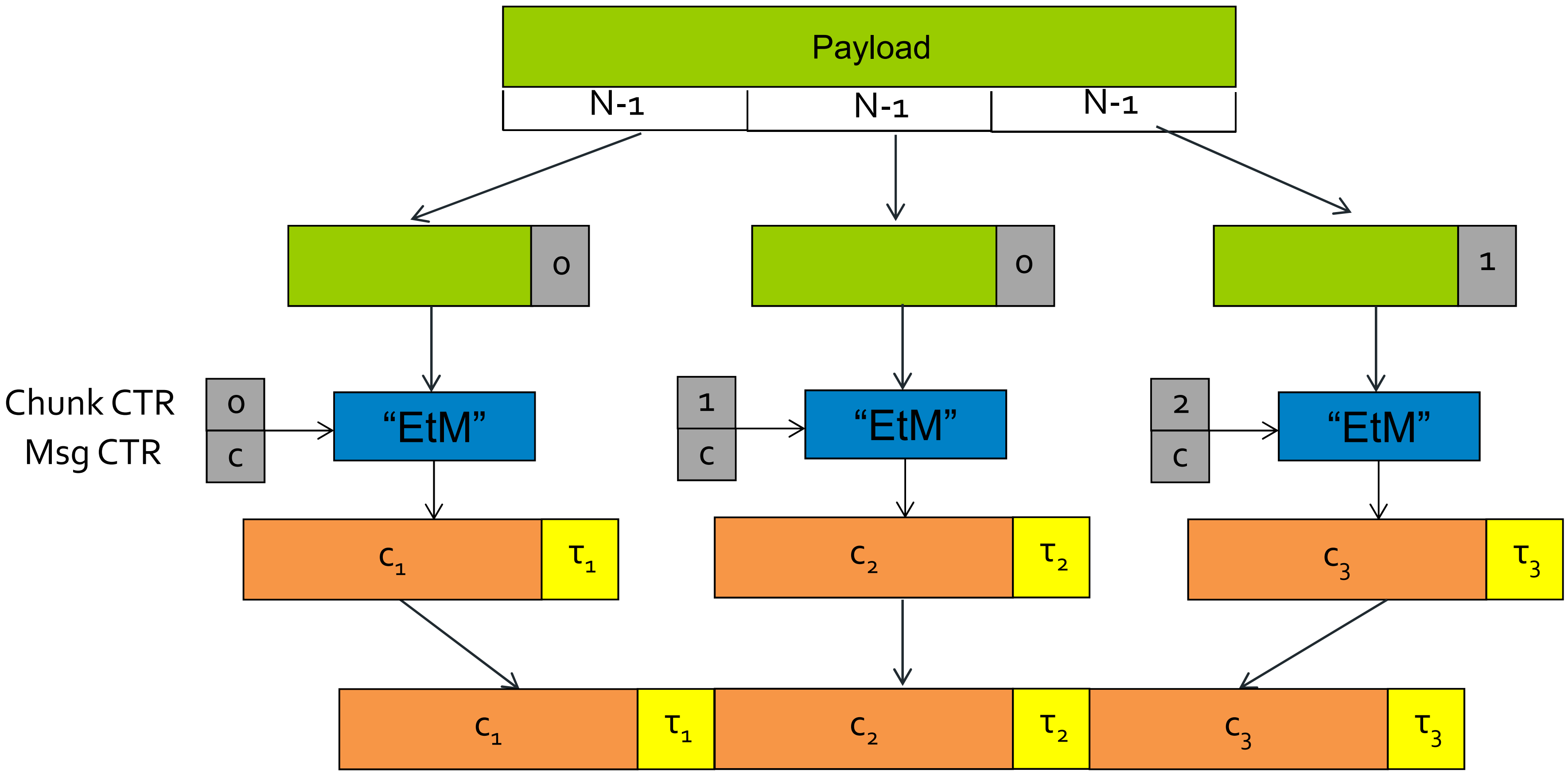
InterMAC

InterMAC

- An encryption scheme proposed in [BDPS12].
- Parameterised by a positive integer N (the **chunk length**).
- Satisfies all 5 security notions:

  IND-sfCFA, IND-sfCTF, BH-CPA, BH-sfCFA, (N + |MAC|)-DOS-sfCFA.
- Applies a generic EtM construction to chunks of data, incorporating additional metadata in the MAC computation.
- Simple, easy to analyse construction; advanced security properties are intuitively obvious.
- Small N: good DoS protection, but larger bandwidth overhead.
- **Idea**: refine and implement InterMAC in OpenSSH to obtain stronger security than is currently available.

InterMAC



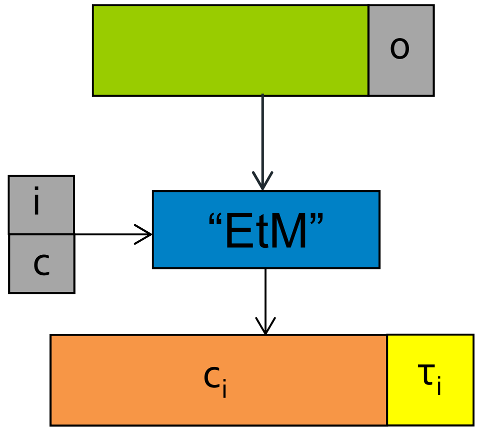
Payload
N-1
N-1
N-1
0
0
1
Chunk CTR
Msg CTR
0
c
"EtM"
1
c
"EtM"
2
c
"EtM"
c1
τ1
c2
τ2
c3
τ3
c1
τ1
c2
τ2
c3
τ3

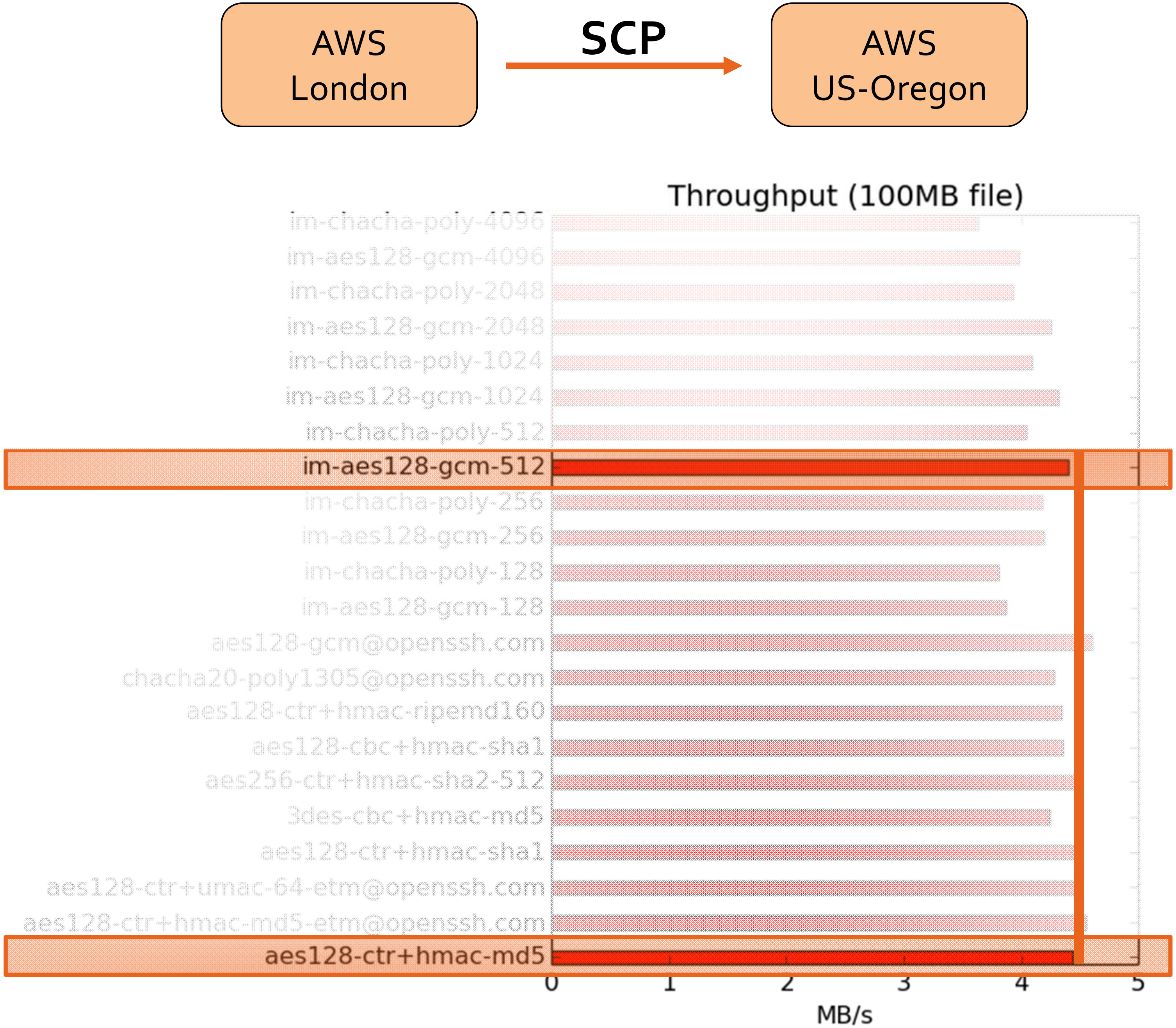
InterMAC: From Theory to Practice

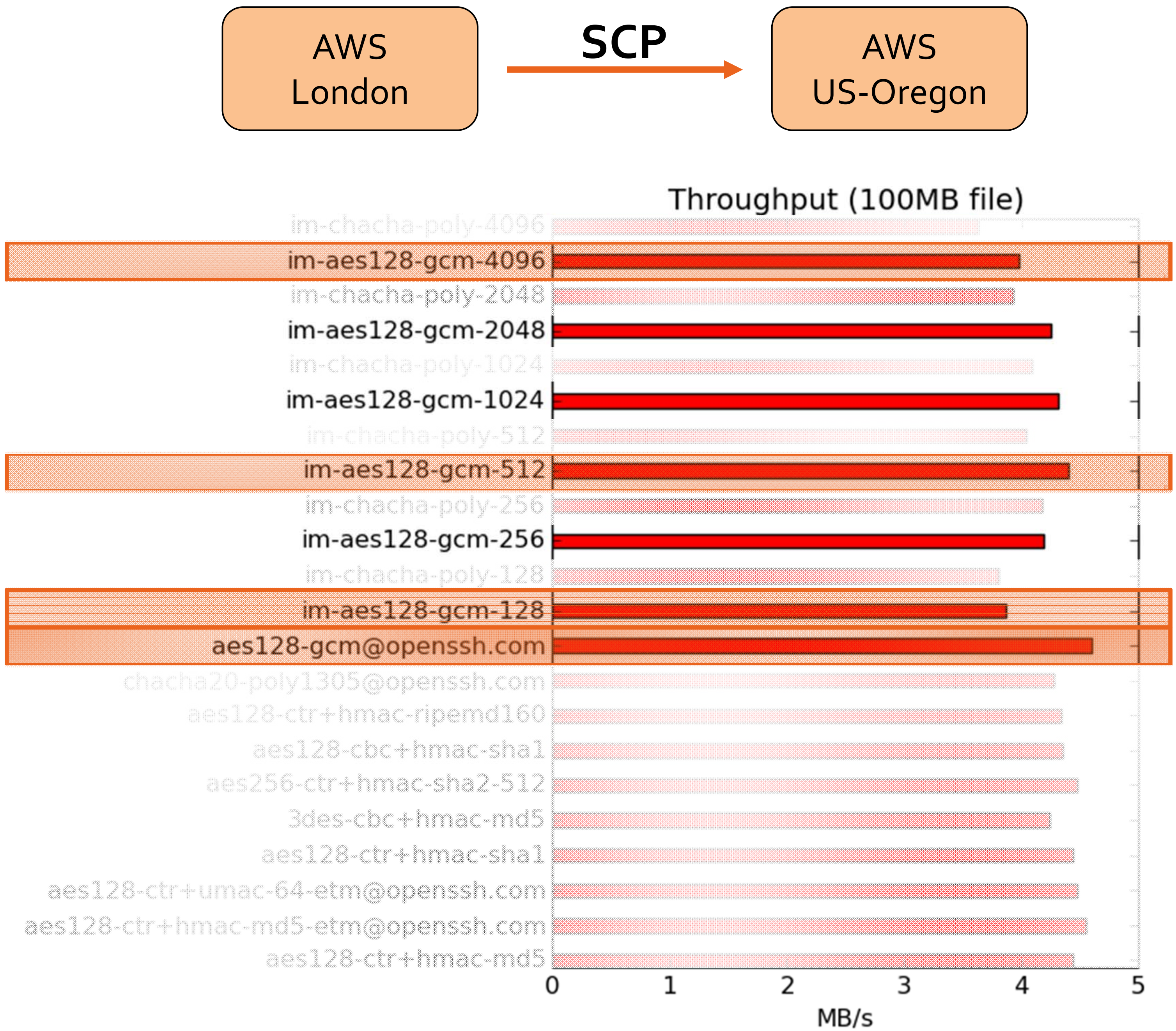
- Use byte-oriented rather than bit-oriented format.
- Abandon underlying SSH packet format (so no length field, no padding byte, no random padding).
- Need some kind of plaintext padding (length not usually a multiple of N-1!): variant of ABYTE padding.
- Replace EtM with nonce-based AEAD, e.g. AES-GCM or ChaCha20-Poly1305.
- Chunk and message counter then become Associated Data, or are used to construct the nonce.
  - We choose the latter.

InterMAClib and OpenSSH

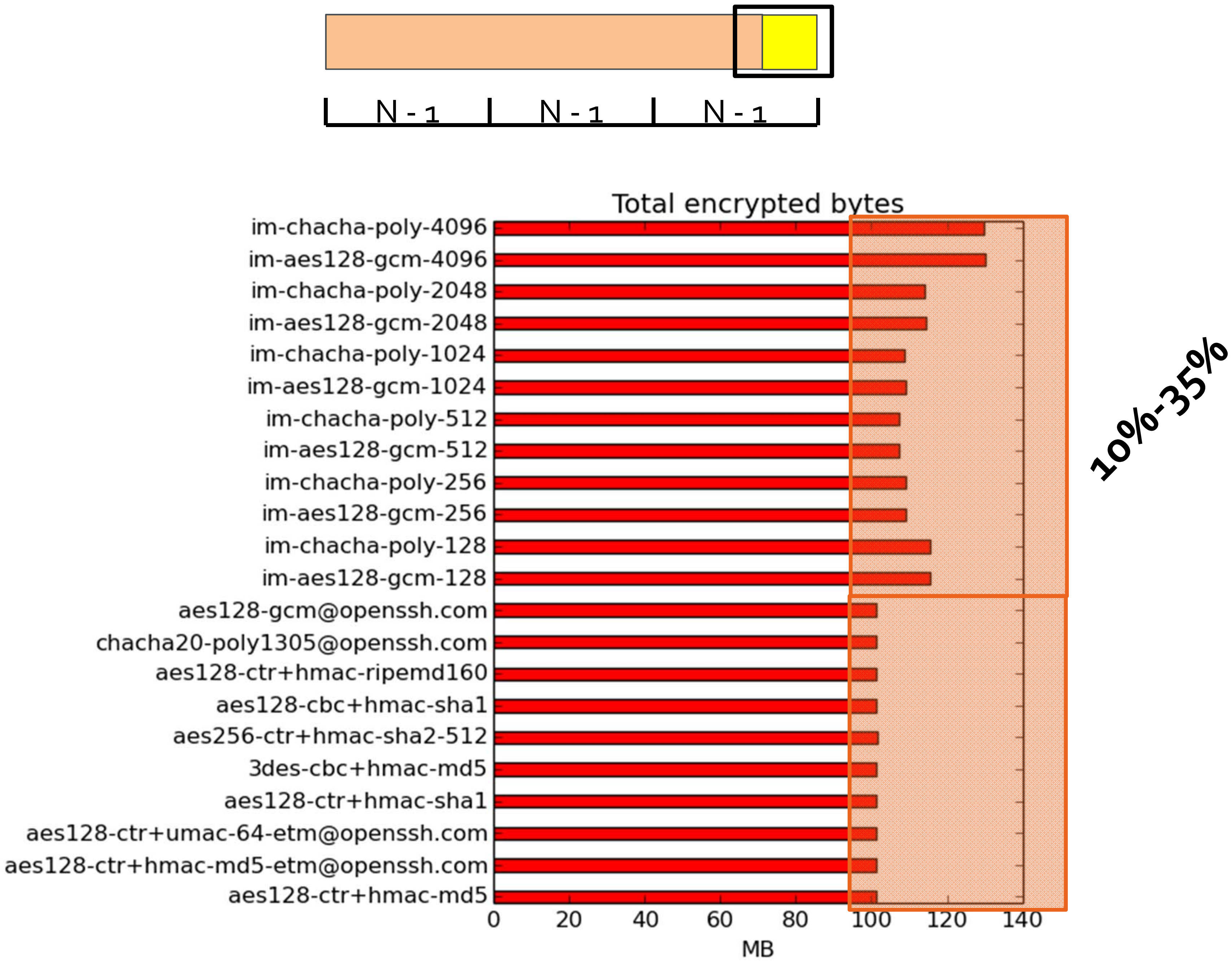
- C-implementation of InterMAC.
- Aim is to make the library easy to use for a developer.
- API: **im_initialise, im_encrypt, im_decrypt**.
- Message counter and nonce management done by the library.
- Currently supports ChaCha-Poly and AES-GCM.
- Easy to extend with other AEAD schemes.
- POC integration into OpenSSH (v7.4).
- SSH InterMAC cipher suites: **im-aes128-gcm-N, im-chacha-poly-N**.




AWS
London
SCP
AWS
US-Oregon
Throughput (100MB file)
im-chacha-poly-4096
im-aes128-gcm-4096
im-chacha-poly-2048
im-aes128-gcm-2048
im-chacha-poly-1024
im-aes128-gcm-1024
im-chacha-poly-512
im-aes128-gcm-512
im-chacha-poly-256
im-aes128-gcm-256
im-chacha-poly-128
im-aes128-gcm-128
aes128-gcm@openssh.com
chacha20-poly1305@openssh.com
aes128-ctr+hmac-ripemd160
aes128-cbc+hmac-sha1
aes256-ctr+hmac-sha2-512
3des-cbc+hmac-md5
aes128-ctr+hmac-sha1
aes128-ctr+umac-64-etm@openssh.com
aes128-ctr+hmac-md5-etm@openssh.com
aes128-ctr+hmac-md5
0
1
2
3
4
5
MB/s



AWS
London
SCP
AWS
US-Oregon
Throughput (100MB file)
im-chacha-poly-4096
im-aes128-gcm-4096
im-chacha-poly-2048
im-aes128-gcm-2048
im-chacha-poly-1024
im-aes128-gcm-1024
im-chacha-poly-512
im-aes128-gcm-512
im-chacha-poly-256
im-aes128-gcm-256
im-chacha-poly-128
im-aes128-gcm-128
aes128-gcm@openssh.com
chacha20-poly1305@openssh.com
aes128-ctr+hmac-ripemd160
aes128-cbc+hmac-sha1
aes256-ctr+hmac-sha2-512
3des-cbc+hmac-md5
aes128-ctr+hmac-sha1
aes128-ctr+umac-64-etm@openssh.com
aes128-ctr+hmac-md5-etm@openssh.com
aes128-ctr+hmac-md5
0
1
2
3
4
5
MB/s



N - 1
N - 1
N - 1
Total encrypted bytes
im-chacha-poly-4096
im-aes128-gcm-4096
im-chacha-poly-2048
im-aes128-gcm-2048
im-chacha-poly-1024
im-aes128-gcm-1024
im-chacha-poly-512
im-aes128-gcm-512
im-chacha-poly-256
im-aes128-gcm-256
im-chacha-poly-128
im-aes128-gcm-128
aes128-gcm@openssh.com
chacha20-poly1305@openssh.com
aes128-ctr+hmac-ripemd160
aes128-cbc+hmac-sha1
aes256-ctr+hmac-sha2-512
3des-cbc+hmac-md5
aes128-ctr+hmac-sha1
aes128-ctr+umac-64-etm@openssh.com
aes128-ctr+hmac-md5-etm@openssh.com
aes128-ctr+hmac-md5
0
20
40
60
80
100
120
140
MB
10%-35%

Concluding Remarks

Concluding Remarks

- We have developed a deeper understanding of the diverse set of encryption modes available in (Open)SSH.
    - Measurement study, new attacks on CBC mode, security analysis
- Formal modelling of security for the goals targeted by SSH.
- None of the schemes in use possesses all the security properties desirable for SSH.
    - Boundary-hiding and DoS-resistance not achieved.
- Yet such schemes do exist, e.g. InterMAC from [BDPS12].
- In our current work, we are developing and prototyping efficient, provably secure alternatives that have all the desired properties.