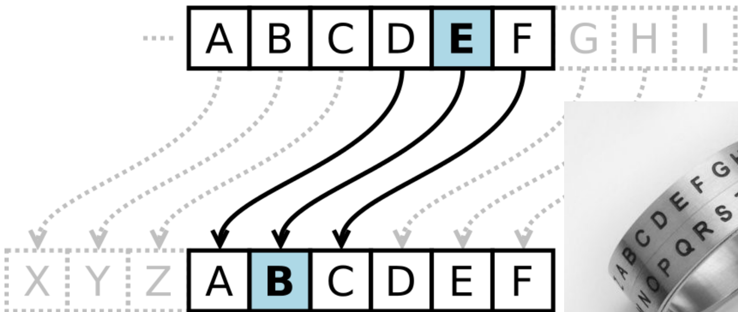
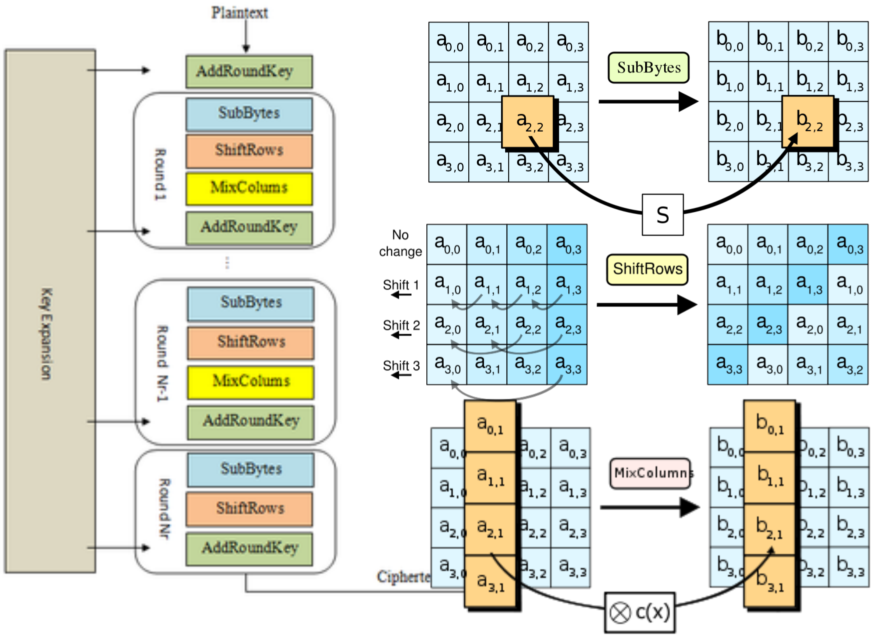
Lecture 8: Secure Channels

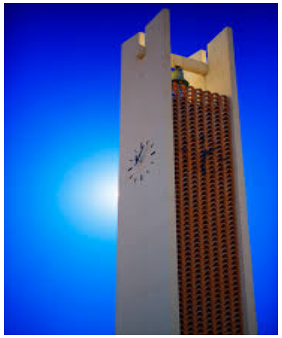
CS 181S

October 1, 2018

Crypto Thus Far…



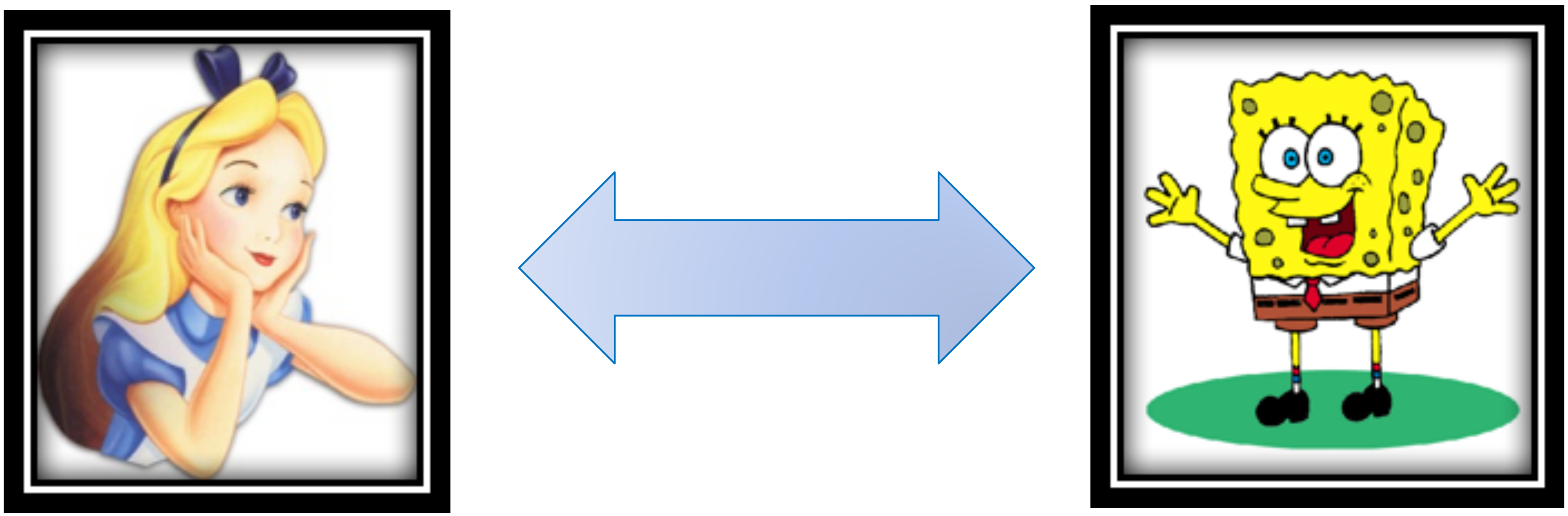




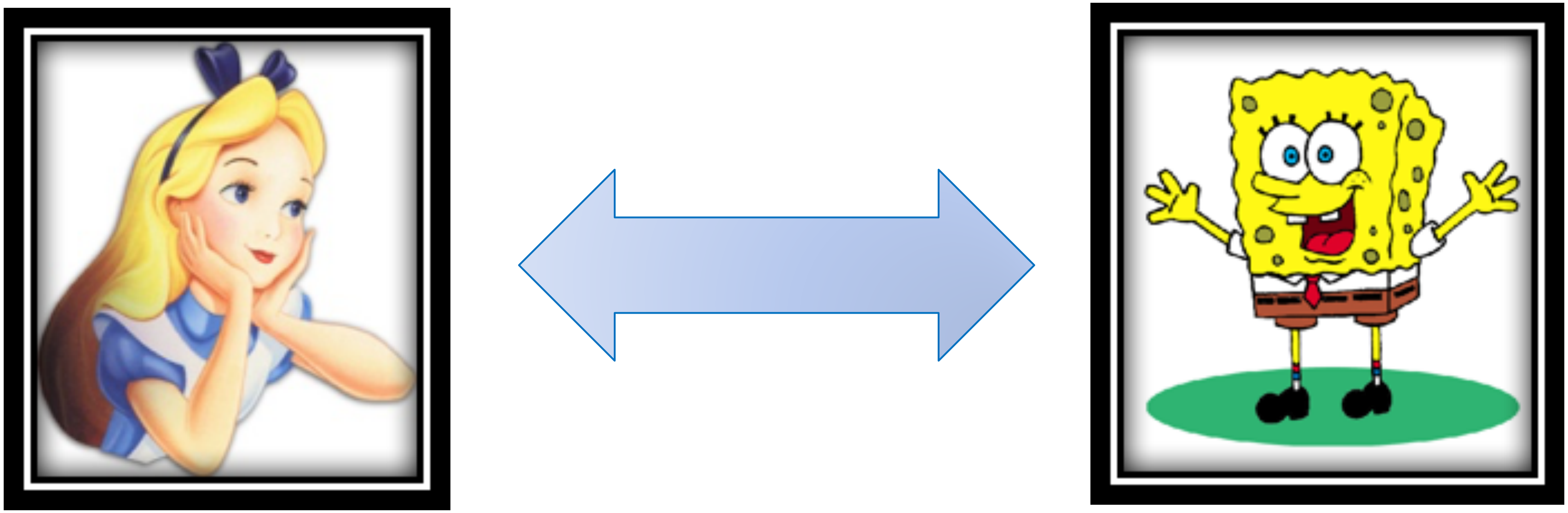
Today: Secure Channels

- **Threat:** attacker who controls the network
  - Dolev-Yao model: attacker can read, modify, delete messages
- **Harm:** conversation can be learned (violating confidentiality) or changed (violating integrity) by attacker
- **Vulnerability:** communication channel between sender and receiver can be controlled by other principals
- **Countermeasure:** all the crypto we've seen so far…

Today: Secure Channels



Today: Secure Channels



Requirements:

1) Channel must provide both confidentiality and integrity
2) A and B must agree on session key(s)
3) A and B must agree on cipher suite (crypto protocols, encryption mode, key lengths)
4) Must detecting missing messages & replay attacks.
5) Must maintain connection (and be able to end it)

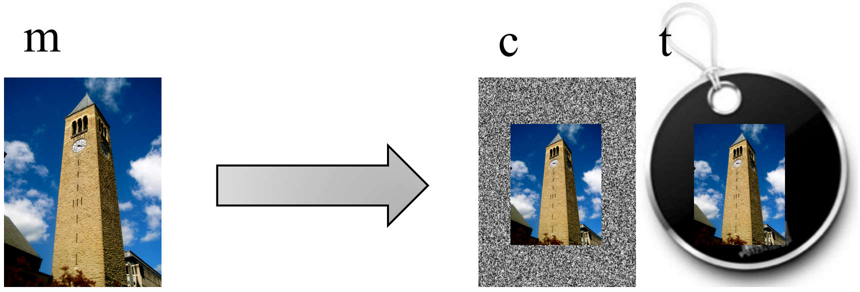
Encrypt and MAC

- **Pro:** can compute Enc and MAC in parallel
- **Con:** MAC must protect confidentiality
  (not actually a requirement we ever stipulated)

- Example: **`ssh`** (Secure Shell) protocol
  - recommends AES-128-CBC for encryption
  - recommends HMAC with SHA-2 for MAC

Encrypt and MAC

```
0. k_E = Gen_E(len)
   k_M = Gen_M(len)
1. A: c = Enc(m; k_E)
      t = MAC(m; k_M)
2. A -> B: c, t
3. B: m' = Dec(c; k_E)
      t' = MAC(m'; k_M)
      if t = t'
        then output m'
        else abort
```



Encrypt then MAC

- **Pro:** provably most secure of three options [Bellare & Namprepre 2001]
- **Pro:** don't have to decrypt if MAC fails
  - resist DoS

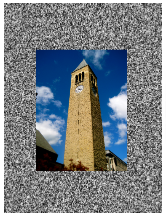
- Example: IPsec (Internet Protocol Security)
  - recommends AES-CBC for encryption and HMAC-SHA2 for MAC, among others
  - or AES-GCM

Encrypt then MAC

m

```
1. A: c = Enc(m; k_E)
      t = MAC(c; k_M)
2. A -> B: c, t
3. B: t' = MAC(c; k_M)
      if t = t'
        then output Dec(c; k_E)
        else abort
```

c



t

MAC then encrypt

- **Pro:** provably next most secure
  - and just as secure as Encrypt-then-MAC for strong enough MAC schemes
  - HMAC and CBC-MAC are strong enough

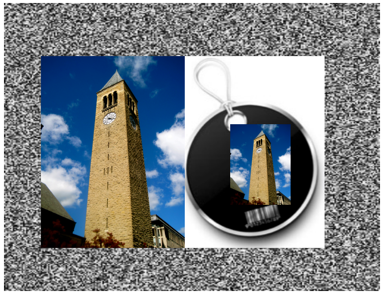
- Example: SSL (Secure Sockets Layer)
  - Many options for encryption, e.g. AES-128-CBC
  - For MAC, standard is HMAC with many options for hash, e.g. SHA-256

MAC then encrypt

```
1. A: t = MAC(m; k_M)
      c = Enc(m,t; k_E)
2. A -> B: c
3. B: m',t' = Dec(c; k_E)
      if t' = MAC(m'; k_M)
         then output m'
         else abort
```

m

c



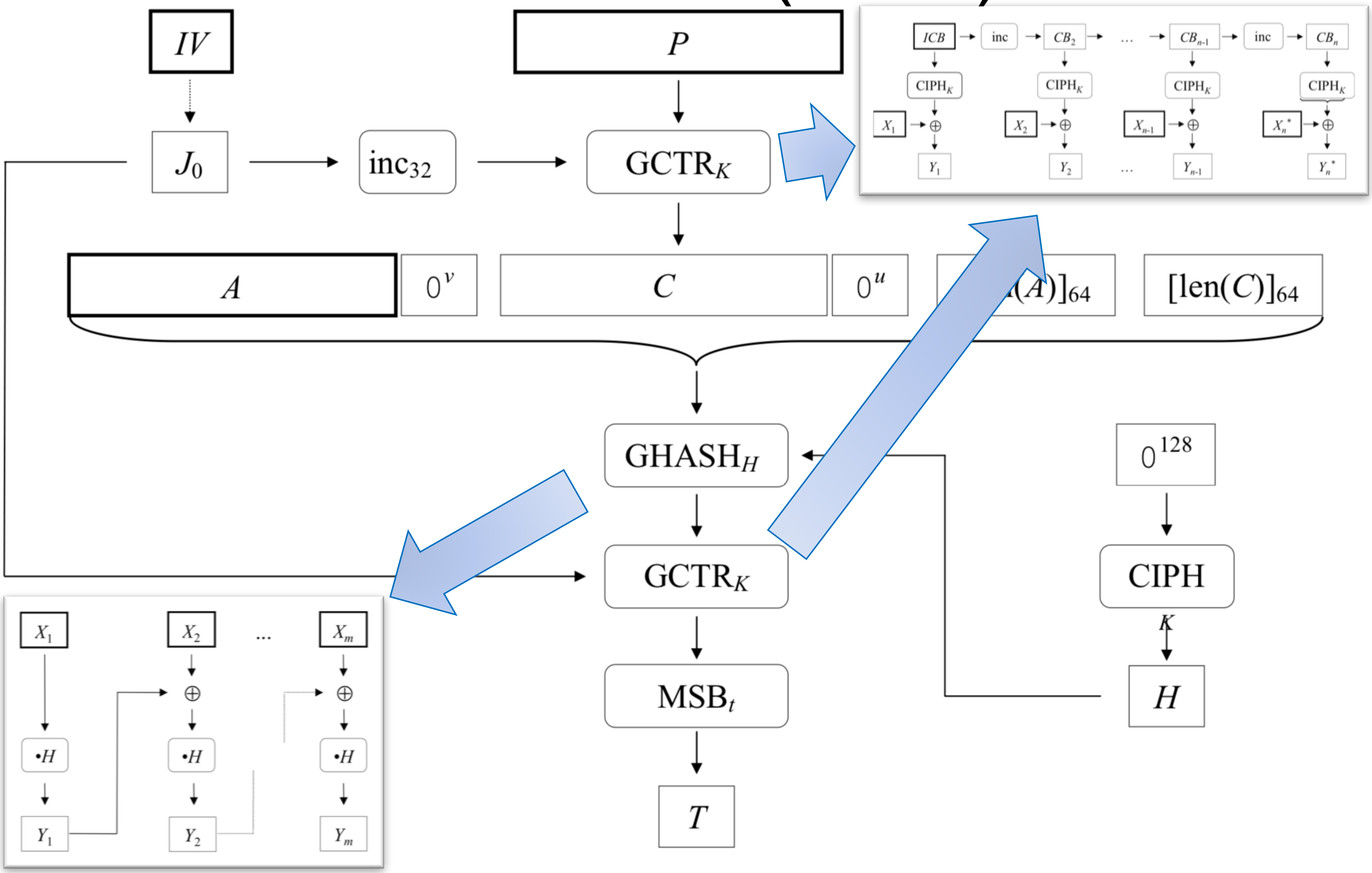
Authenticated encryption

- Three combinations:
  - Enc and MAC
  - Enc then MAC
  - MAC then Enc
- Let's unify all with a pair of algorithms:
  - AuthEnc(m; kE; kM): produce an authenticated ciphertext x of message m under encryption key kEand MAC key kM
  - AuthDec(x; kE; kM): recover the plaintext message m from authenticated ciphertext x, and verify that the MAC is valid, using kE and kM
    - Abort if MAC is invalid

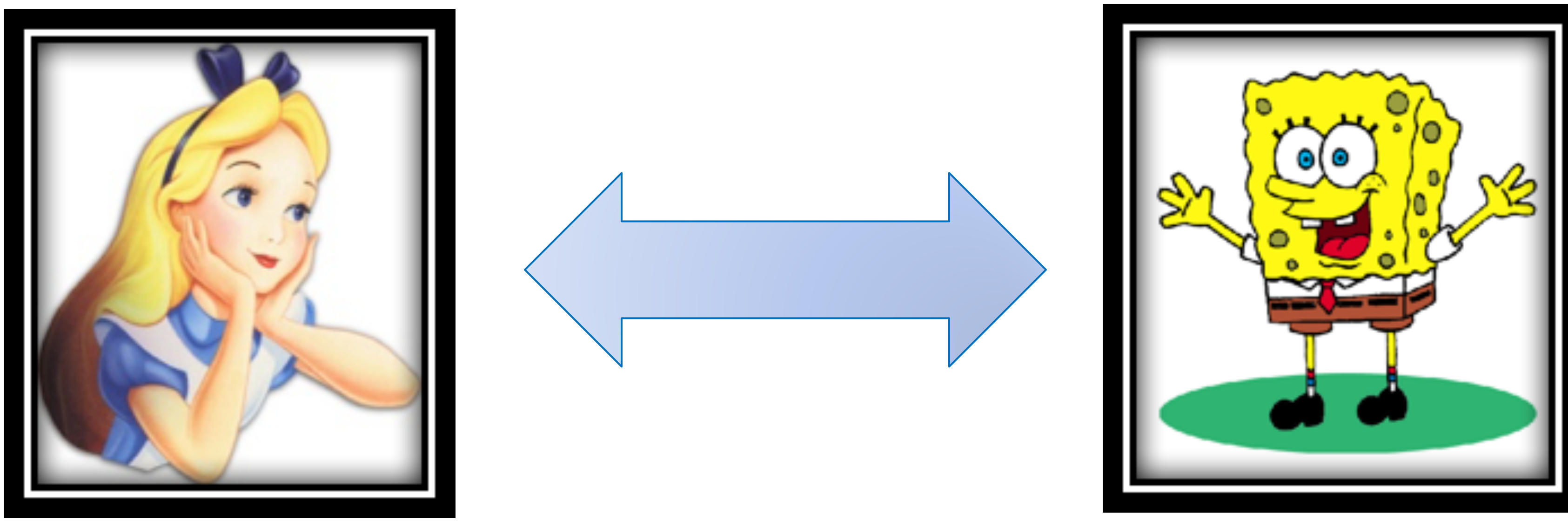
Authenticated encryption

- Newer block cipher modes designed to provide confidentiality and integrity
  - **OCB:** Offset Codebook Mode
  - **CCM:** Counter with CBC-MAC Mode
  - **GCM:** Galois Counter Mode

Galois Counter Mode (GCM)



Today: Secure Channels



Requirements:

1) Channel must provide both confidentiality and integrity
2) A and B must agree on session key(s)
3) A and B must agree on cipher suite (crypto protocols, encryption mode, key lengths)
4) Must detecting missing messages & replay attacks.
5) Must maintain connection (and be able to end it)

Agreeing on a session key

Hybrid Encryption (RSA)

Diffie-Hellman

- `A -> B: g, p, g^a mod p`
- `B -> A: g^b mod p`
- `A,B: k_s := g^ab mod p`

- DH, ECDH

Aside: Key reuse

- Never use same key for both encryption and MAC schemes
- **Principle:** every key in system should have unique purpose

Key derivation

- Have one key: k_s
- Need four keys:
  1. kea: Encrypt Alice to Bob
  2. keb: Encrypt Bob to Alice
  3. kma: MAC Alice to Bob
  4. kmb: MAC Bob to Alice
- How to get four out of one: use a cryptographic hash function H to derive keys...
  1. kea = H(k, "Enc Alice to Bob")
  2. keb = H(k, "Enc Bob to Alice")
  3. kma = H(k, "MAC Alice to Bob")
  4. kmb = H(k, "MAC Bob to Alice")

Key derivation

- Why hash?
  - Destroys any structure in input
  - Produces a fixed-size output that can be truncated, as necessary, to produce key for underlying algorithm
  - Unlikely to ever cause any of four keys to collide
  - Even if one of four keys ever leaks, hard to invert hash to recover k and learn the other keys
- Small problem: maybe the output of H isn't compatible with the output of Gen
  - For most block ciphers and MACs, not a problem
    - they happily take any uniformly random sequence of bits of the right length as keys
  - For DES, it is a problem
    - has weak keys that Gen should reject
  - For many asymmetric algorithms, it would be a problem
    - keys have to satisfy certain algebraic properties

Today: Secure Channels

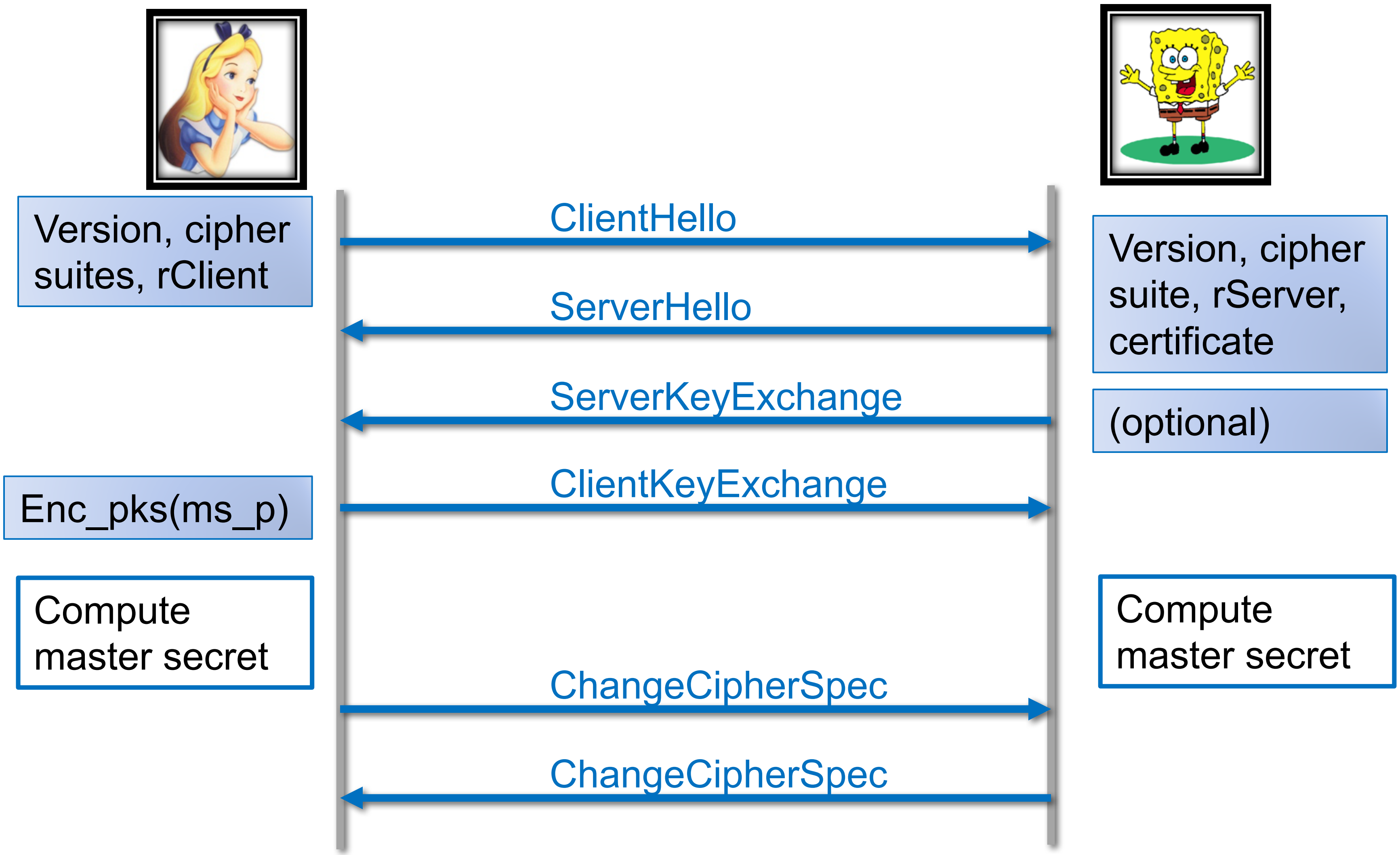
Requirements:

1) Channel must provide both confidentiality and integrity
2) A and B must agree on session key(s)
3) A and B must agree on cipher suite (crypto protocols, encryption mode, key lengths)
4) Must detecting missing messages & replay attacks.
5) Must maintain connection (and be able to end it)

Secure Socket Layer (SSL)

- SSL 2.0 (1995): designed by Netscape, contains a number of security flaws, prohibited since 2011
- SSL 3.0 (1996): complete re-design, all accepted cipher suites now have known vulnerabilities, prohibited since 2015
- TLS 1.0 (1999): contains known vulnerabilities, suggested migration by June 2018
- TLS 1.1 (2006): update with significant changes in how IVs/padding are handled to prevent known attacks
- TLS 1.2 (2008): update with modern cipher suites
- TLS 1.3 (2018): drops insecure features and introduces additional cipher suites

SSL/TLS Handshake



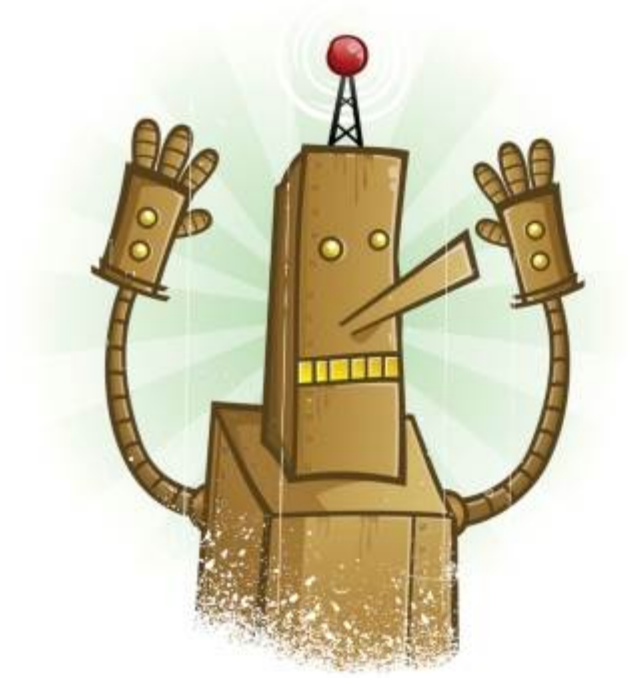
Supported Cipher Suites

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
RSA	Yes	Yes	Yes	Yes	Yes	No
DH-RSA	No	Yes	Yes	Yes	Yes	No
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes
ECDH-RSA	No	No	Yes	Yes	Yes	No
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes
DH-DSS	No	Yes	Yes	Yes	Yes	No
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[42]
ECDH-ECDSA	No	No	Yes	Yes	Yes	No
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes

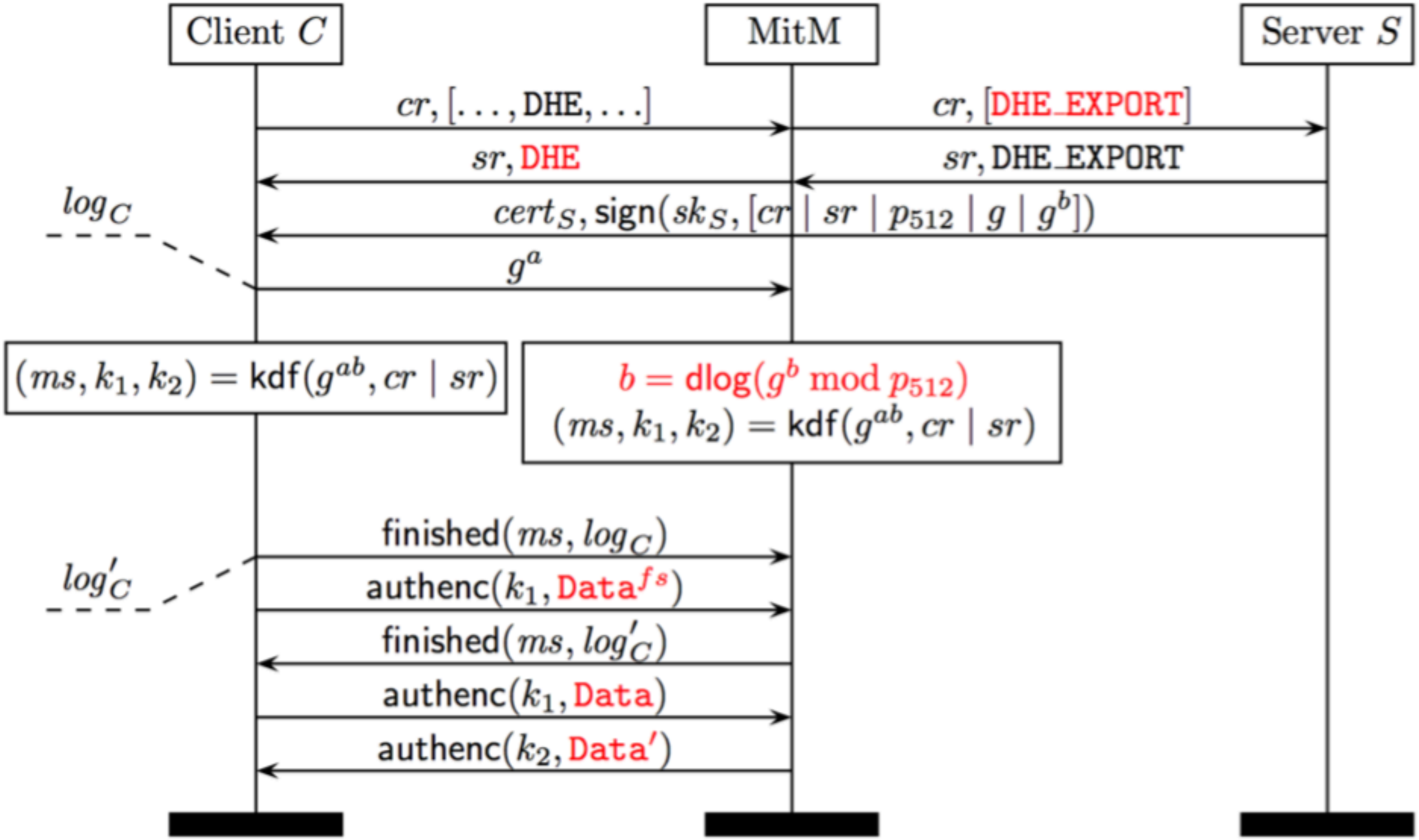
Cipher			Protocol version					
Type	Algorithm	Nominal strength (bits)	SSL 2.0	SSL 3.0 [n 1][n 2][n 3][n 4]	TLS 1.0 [n 1][n 3]	TLS 1.1 [n 1]	TLS 1.2 [n 1]	TLS 1.3
Block cipher with mode of operation	AES GCM[44][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	Secure
	AES CCM[45][n 5]		N/A	N/A	N/A	N/A	Secure	Secure
	AES CBC[n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
	Camellia GCM[46][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A
	Camellia CBC[47][n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
	ARIA GCM[48][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A
	ARIA CBC[48][n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
	SEED CBC[49][n 6]	128	N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
	3DES EDE CBC[n 6][n 7]	112[n 8]	Insecure	Insecure	Insecure	Insecure	Insecure	N/A
	GOST 28147-89 CNT[43][n 7]	256	N/A	N/A	Insecure	Insecure	Insecure	N/A
	IDEA CBC[n 6][n 7][n 9]	128	Insecure	Insecure	Insecure	Insecure	N/A	N/A
	DES CBC[n 6][n 7][n 9]	56	Insecure	Insecure	Insecure	Insecure	N/A	N/A
		40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A
	RC2 CBC[n 6][n 7]	40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A
Stream cipher	ChaCha20-Poly1305[54][n 5]	256	N/A	N/A	N/A	N/A	Secure	Secure
	RC4[n 11]	128	Insecure	Insecure	Insecure	Insecure	Insecure	N/A
		40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A

Padding Oracle On Downgraded Legacy Encryption (POODLE)

Return of Beichenbacher's Oracle Threat (ROBOT)



Logjam



Client C
MitM
Server S
$cr, [\ldots, \text{DHE}, \ldots]$
cr, [DHE_EXPORT]
sr, DHE
sr, DHE_EXPORT
$cert_S, \mathsf{sign}(sk_S, [cr \mid sr \mid p_{512} \mid g \mid g^b])$
log_C
g^a
$(ms, k_1, k_2) = \mathsf{kdf}(g^{ab}, cr \mid sr)$
$b = \mathsf{dlog}(g^b \bmod p_{512})$
$(ms, k_1, k_2) = \mathsf{kdf}(g^{ab}, cr \mid sr)$
finished(ms, log_C)
log'_C
authenc(k_1, Data^{fs})
finished(ms, log'_C)
authenc(k_1, Data)
authenc(k_2, Data')

Today: Secure Channels

Requirements:

1) Channel must provide both confidentiality and integrity
2) A and B must agree on session key(s)
3) A and B must agree on cipher suite (crypto protocols, encryption mode, key lengths)
4) Must detecting missing messages & replay attacks.
5) Must maintain connection (and be able to end it)

Message numbers

- Aka sequence numbers
- Every message that Alice sends is numbered
  - 1, 2, 3, ...
  - numbers increase monotonically
  - never reuse a number
- Bob keeps state to remember last message number he received
- Bob accepts only increasing message numbers
- And ditto all the above, for Bob sending to Alice
  - so each principal keeps two independent counters: messages sent, messages received

Message numbers

What if Bob detects a gap? e.g. 1, 2, 5

- Maybe Mallory deleted messages 3 and 4 from network
- Maybe Mallory detectably changed 3 and 4, causing Bob to discard them
- In either case, channel is under active attack
  - Absent availability goals, time to PANIC: abort protocol, produce appropriate information for later auditing, shut down channel

What if network non-maliciously dropped messages or will deliver them later?

- Let's assume underlying transport protocol guarantees that won't happen (e.g. TCP)

Message numbers

- Message number usually implemented as a fixed-size unsigned integer, e.g., 32 or 48 or 64 bits
- What if that `int` overflows and wraps back around to 0?
  - Message number **must** be unique within conversation to prevent Mallory from replaying old conversation
  - So conversation **must** stop at that point
  - Can start a new conversation with a new session key

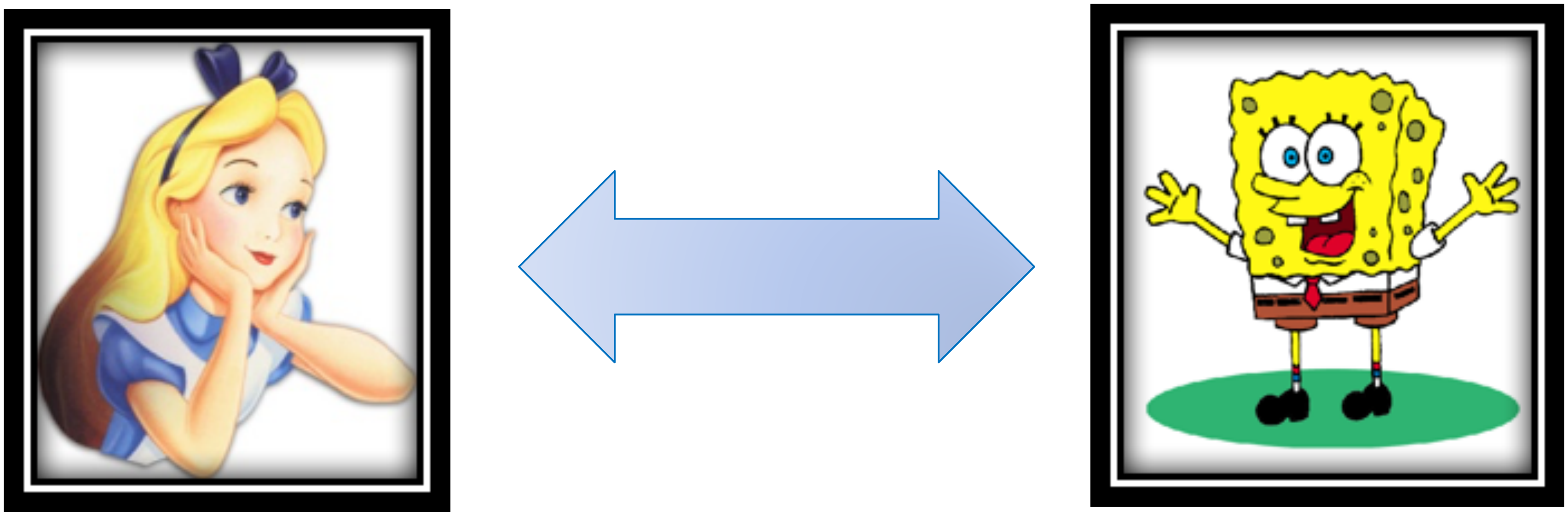
To send a message from A to B

```
1. A:
      increment sent_ctr;
      if sent_ctr overflows then abort;
      x = AuthEnc(sent_ctr, m; kea; kma)
2. A -> B: x
3. B:
      i,m = AuthDec(x; kea; kma);
      increment rcvd_ctr;
      if i != rcvd_ctr then abort;
      output m
```

To send a message from B to A

```
1.   :
       increment sent_ctr;
       if sent_ctr overflows then abort;
       x = AuthEnc(sent_ctr, m; keb; kmb)
2. B -> A: x
3. A:
       i,m = AuthDec(x; keb; kmb);
       increment rcvd_ctr;
       if i != rcvd_ctr then abort;
       output m
```

Today: Secure Channels



Requirements:

1) Channel must provide both confidentiality and integrity
2) A and B must agree on session key(s)
3) A and B must agree on cipher suite (crypto protocols, encryption mode, key lengths)
4) Must detecting missing messages & replay attacks.
5) Must maintain connection (and be able to end it)

TLS record

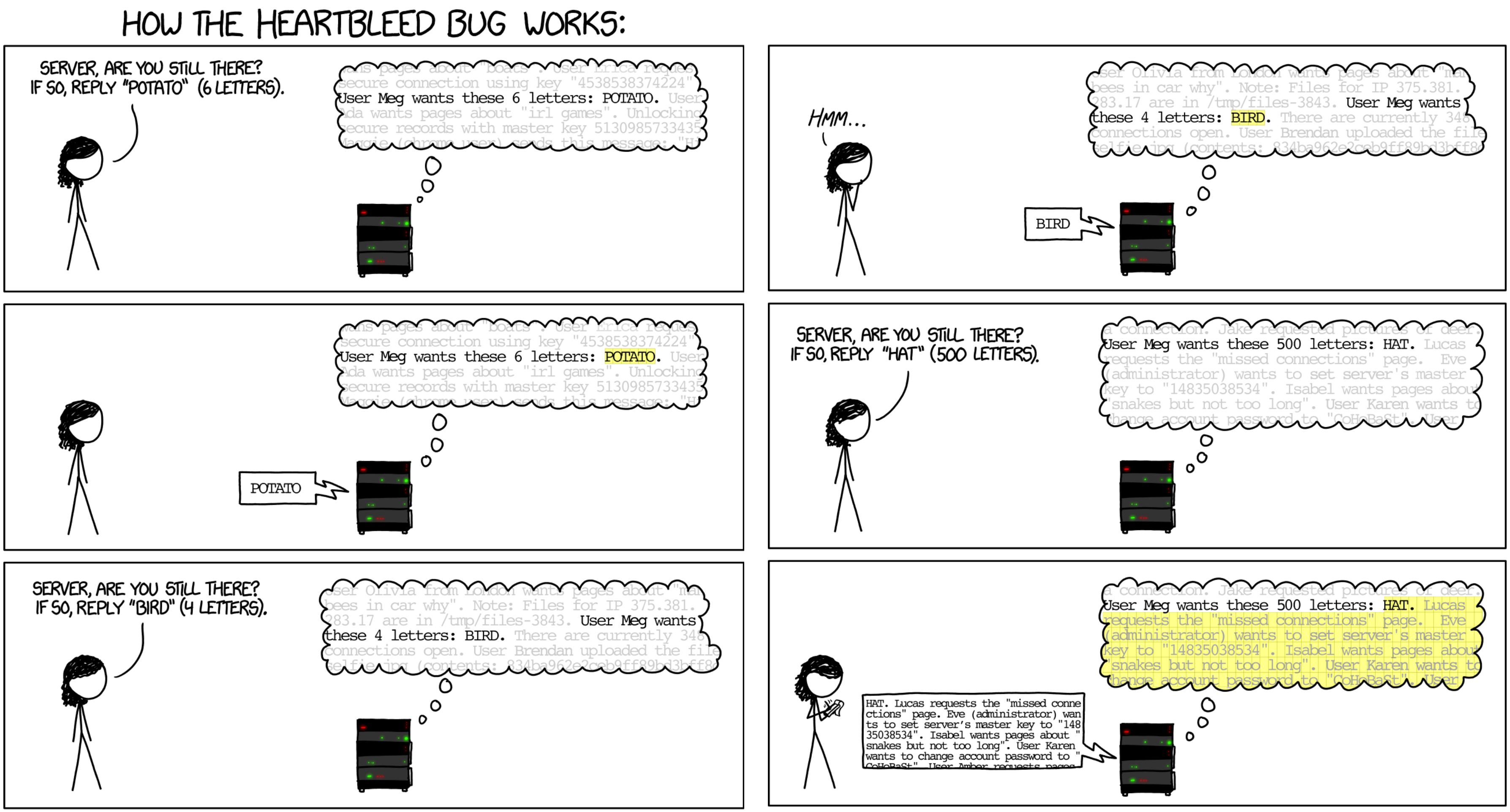
<table>
<tr><th>+</th><th>Byte +0</th><th>Byte +1</th><th>Byte +2</th><th>Byte +3</th></tr>
<tr><td>Byte
0</td><td>Content type</td><td colspan="3"></td></tr>
<tr><td rowspan="2">Bytes
1..4</td><td colspan="2">Version</td><td colspan="2">Length</td></tr>
<tr><td>(Major)</td><td>(Minor)</td><td>(bits 15..8)</td><td>(bits 7..0)</td></tr>
<tr><td>Bytes
5..(m–1)</td><td colspan="4">Protocol message(s)</td></tr>
<tr><td>Bytes
m..(p–1)</td><td colspan="4">MAC (optional)</td></tr>
<tr><td>Bytes
p..(q–1)</td><td colspan="4">Padding (block ciphers only)</td></tr>
</table>

Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

+	Byte +0		2	Byte +3
Byte 0	Content type			
Bytes 1..4	Versi		Length	
	(Major)		*8)*	*(bits 7..0)*
Bytes 5..(*m*–1)				
Bytes *m*..(*p*–1)				
Bytes *p*..(*q*–1)	Padding (block ciphers only)			

Heartbeat


HOW THE HEARTBLEED BUG WORKS:
SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
User Meg wants these 6 letters: POTATO.
POTATO
SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Meg wants these 4 letters: BIRD.
HMM...
BIRD
SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT.
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "

Heartbleed

Truncation Attack