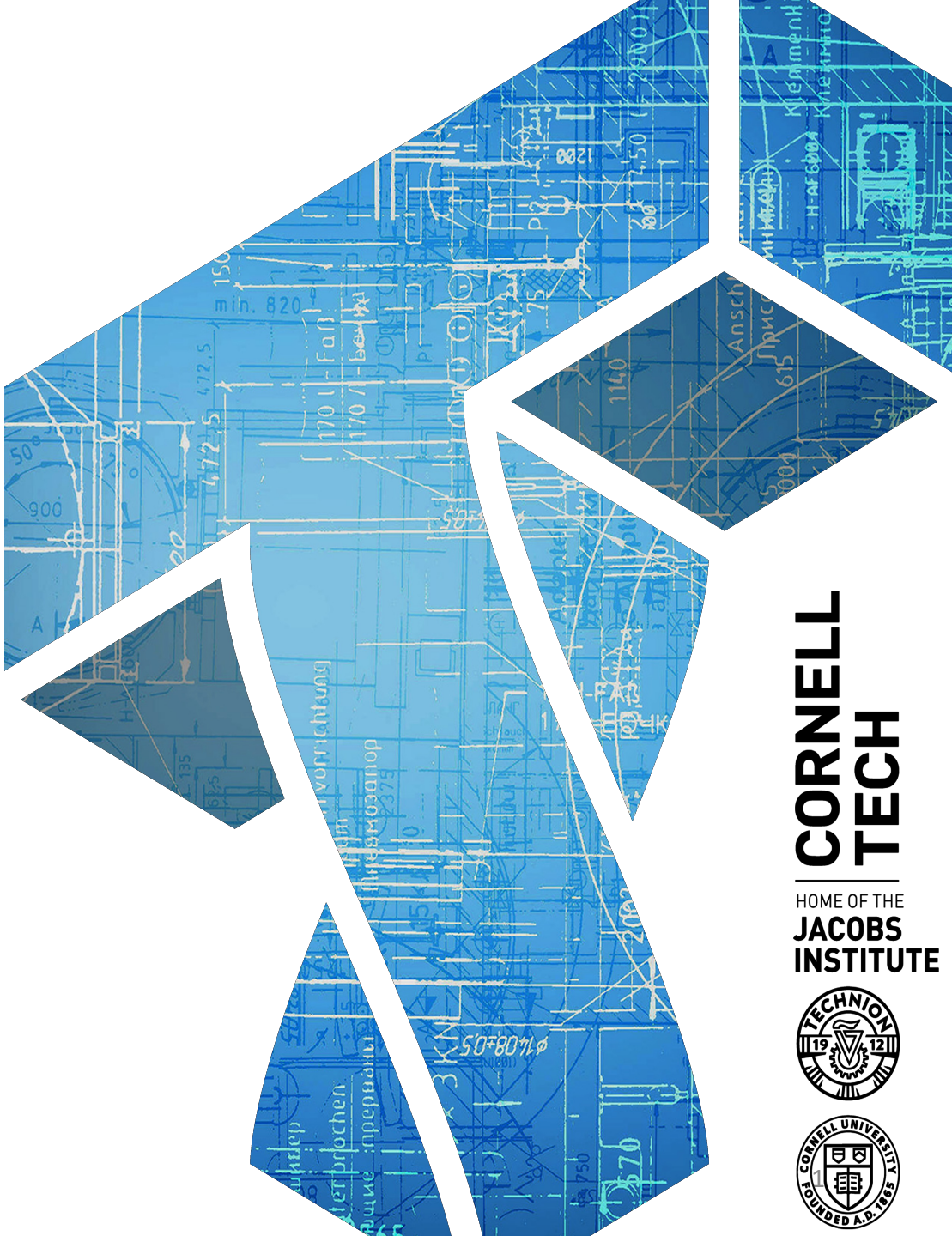
CS 5435: Cryptography

Instructor: Tom Ristenpart

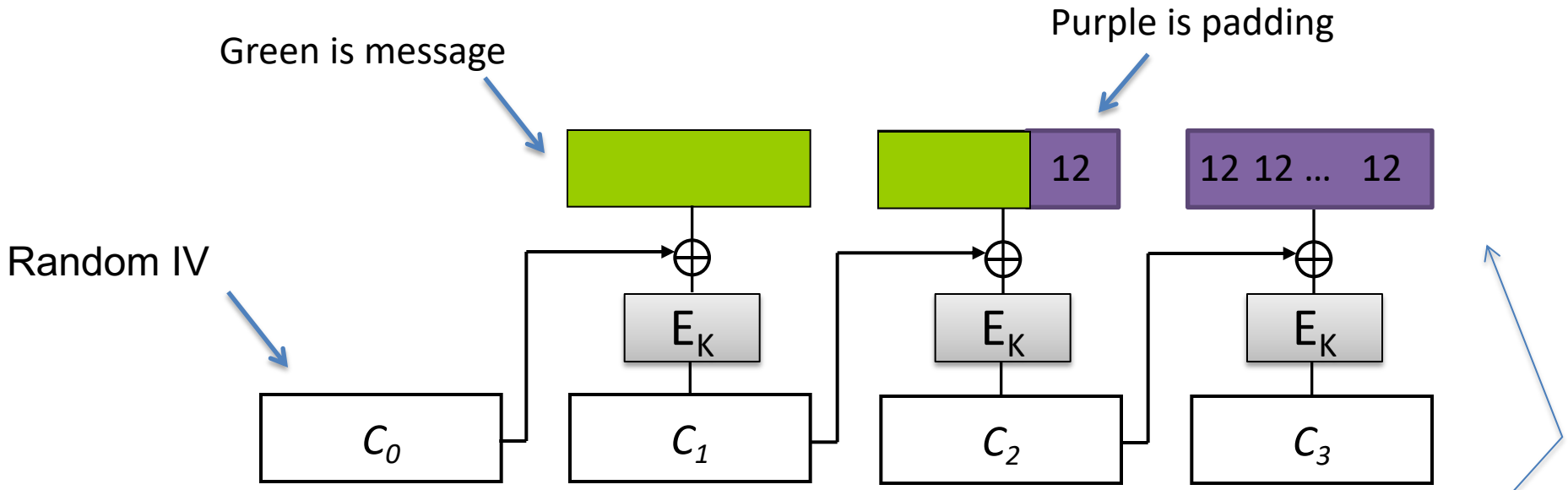
https://github.com/tomrist/cs5435-spring2024



CORNELL TECH

HOME OF THE JACOBS INSTITUTE

PKCS #7 Padding for CBC Mode



Possible paddings:

01 02 02 03 03 03 etc.

"Lengths longer than necessary might be desirable to frustrate attacks on a protocol that are based on analysis of the lengths of exchanged messages."
RFC 5246 → Called "traffic analysis attacks"

PKCS #7 padding oracles

Low byte of M[1] most likely equals i ⊕ 01

Adversary obtains ciphertext C[0],C[1],C[2]
Let R be arbitrary n bits

R, C[0] , C[1] →
← error
R , C[0]⊕1 , C[1] →
← error
R , C[0]⊕2 , C[1] →
← error
…
R , C[0]⊕i , C[1] →
← ok

Dec(K, C)
M'[1] || … || M'[m] = CBC-Dec(K,C)
P = RemoveLastByte(M'[m])
while i < int(P):
    P' = RemoveLastByte(M'[m])
    If P' != P then
        Return error
Return ok

Why? Let X[1] = D(K,C1)

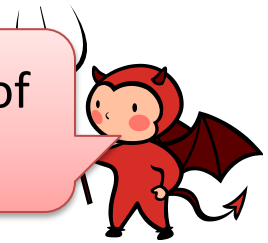
C[0][16] ⊕ X[1][16] = M[1][16]
C[0][16] ⊕ i ⊕ X[1][16] = 01

M[1][16] ⊕ i = 01

Actually, it could be that:
M[1][16] ⊕ i = 02

Implies that M[1][15] = 02
We can rule out with an additional query

PKCS #7 padding oracles



Second lowest byte of M[1] equals i ⊕ 02

Adversary obtains ciphertext C[0],C[1],C[2]
Let R be arbitrary n bits

R, C[0]⊕0||j , C[1] →
← error
R , C[0]⊕1||j , C[1] →
← error
R , C[0]⊕2||j , C[1] →
← error
…
R , C[0]⊕i||j , C[1] →
← ok

Dec(K, C)
M'[1] || … || M'[m] = CBC-Dec(K,C)
P = RemoveLastByte(M'[m])
while i < int(P):
    P' = RemoveLastByte(M'[m])
    If P' != P then
        Return error
Return ok

Set j = M[1][16] ⊕ 01 ⊕ 02

Keep going to recover entire block of message M[1]
Can repeat with other blocks M[2], M[3], …
Worst case: 256*16 queries per block

Can we change decryption implementation?

```
Dec( K, C )
M[1] || … || M[m] = CBC-Dec(K,C)
P = RemoveLastByte(M[m])
while i < int(P):
        P’ = RemoveLastByte(M[m])
        If P’ != P then
                Return error
Return ok
```

“ok” / “error” stand-ins for some other behavior:

- Passing data to application layer (web server)
- Returning other error code (if padding fails)

Chosen ciphertext attacks against CBC

Attack	Description	Year
Vaudenay	10's of chosen ciphertexts, recovers message bits from a ciphertext. Called "padding oracle attack"	2001
Canvel et al.	Shows how to use Vaudenay's ideas against TLS	2003
Degabriele, Paterson	Breaks IPsec encryption-only mode	2006
Albrecht et al.	Plaintext recovery against SSH	2009
Duong, Rizzo	Breaking ASP.net encryption	2011
Jager, S		
Duong,		
AlFarda		
AlFarda		
Albrech		

INFOWORLD TECH WATCH

About | Informed news analysis every weekday

By Ted Samson, InfoWorld | SEP 20, 2010 6:57 AM PST

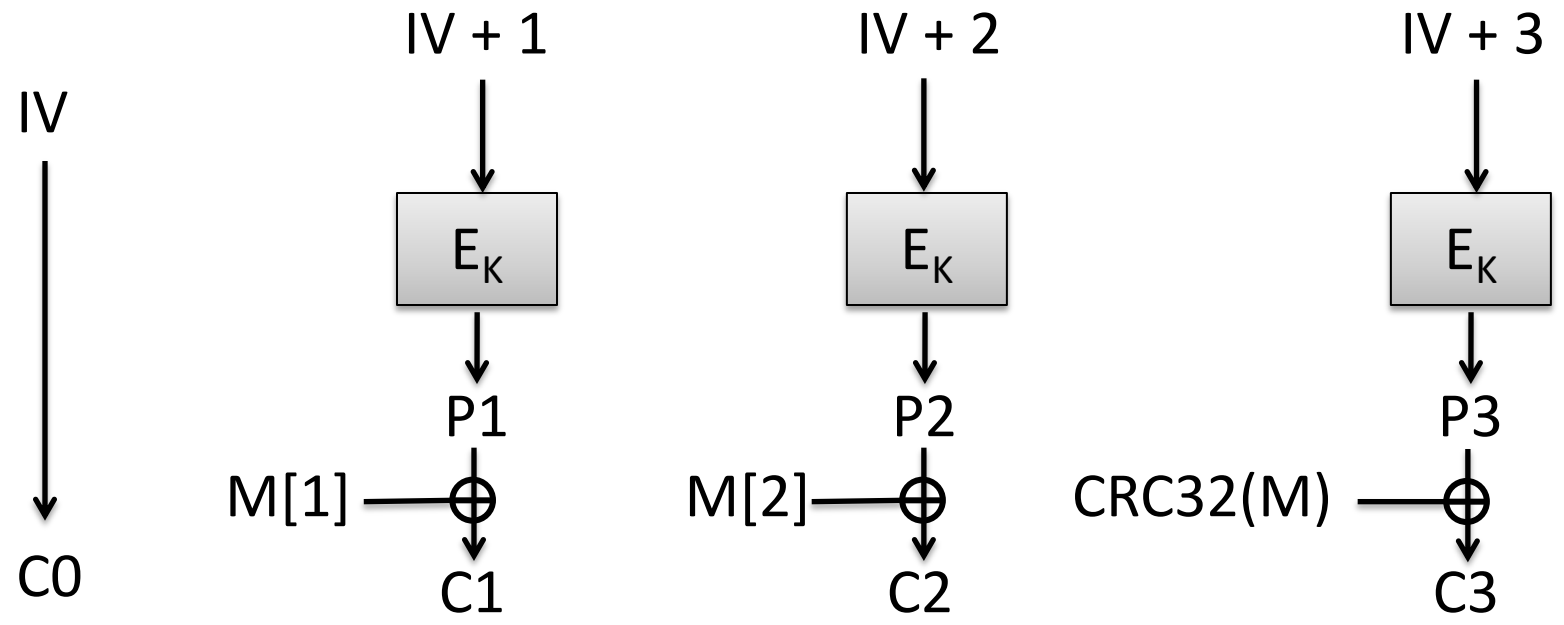
Microsoft rolls out stopgap to ASP.Net vulnerability

Malicious hackers can exploit the vulnerability to swipe sensitive data from servers and clients

Modes like CTR, CBC, ECB are *not* secure for general-purpose encryption

- ECB does not even provide confidentiality
- CTR mode and CBC mode fail in presence of active attacks
  - Cookie example
  - Padding oracle attacks
- Need authentication mechanisms to help prevent chosen-ciphertext attacks

Non-cryptographic checksums?



CRC32(M) outputs a 32-bit error correction code

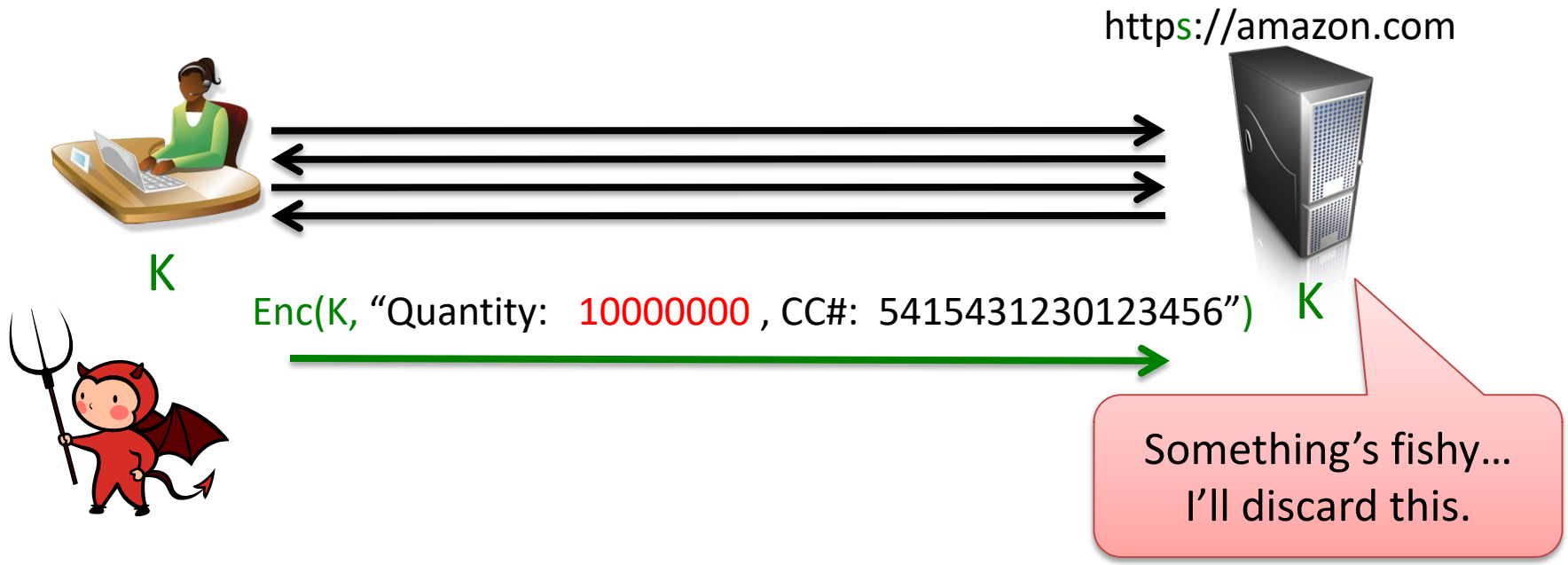
Can simply maul message and CRC32 checksum to ensure correctness

We need new primitives

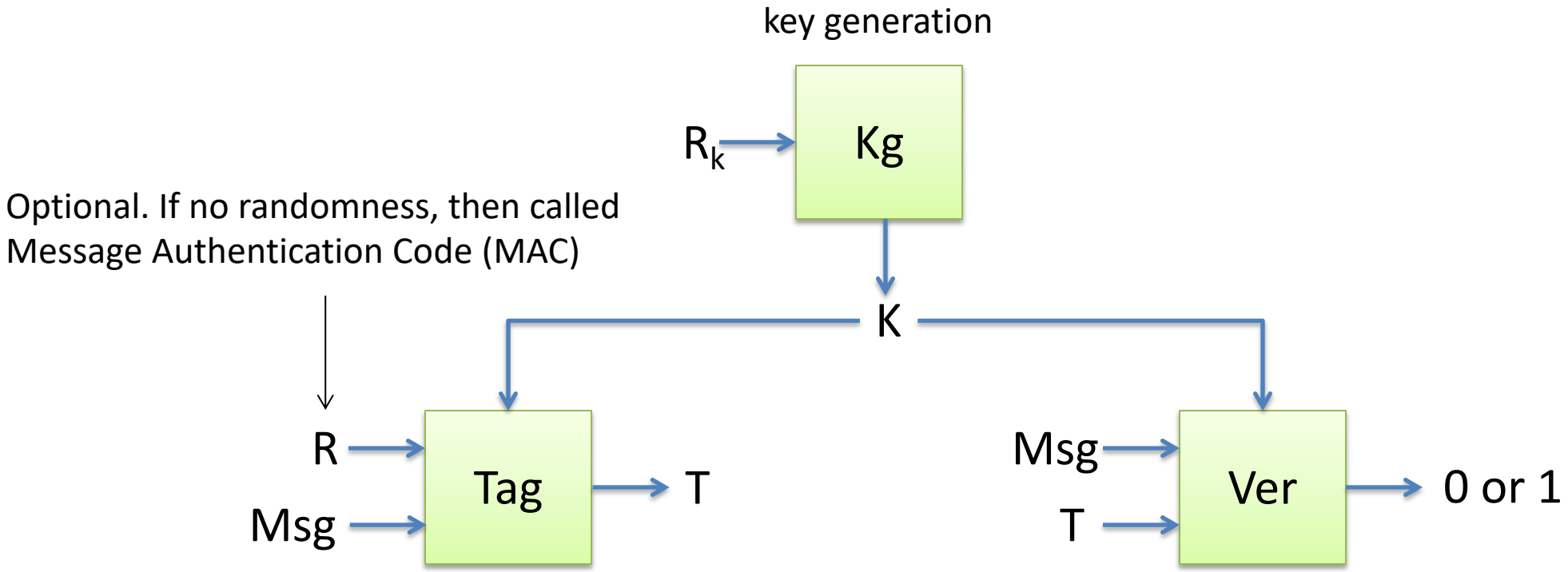
Message authentication makes malicious modifications detectable

Goal is ***authenticated encryption*** (AE): Hide message *and* detect modifications

Can build by combining encryption with message authentication scheme

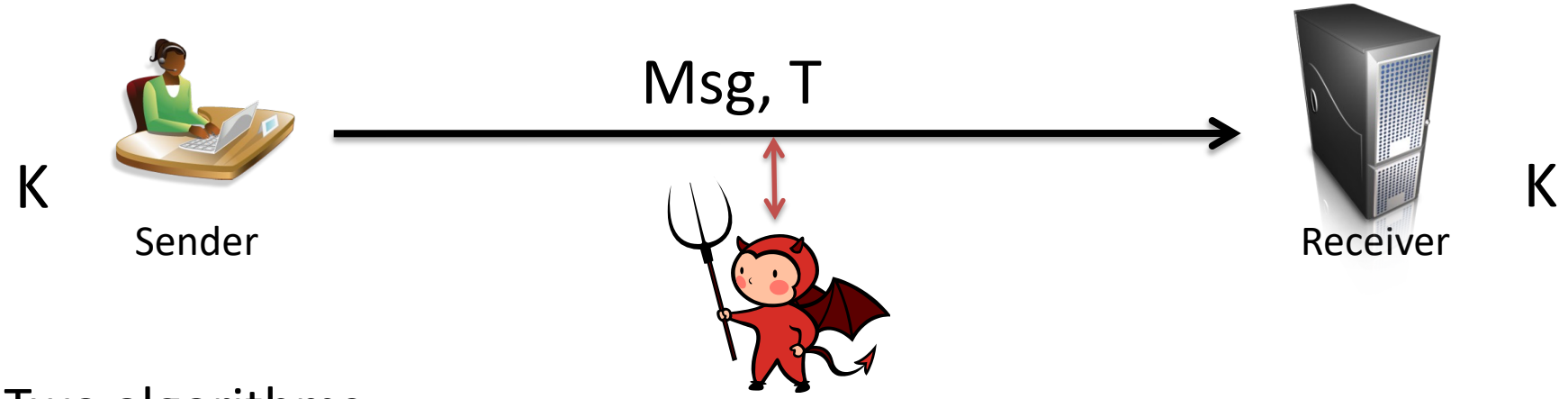


Message authentication



Correctness: Ver(K , Tag(K,Msg,R)) = 1 with probability 1 over randomness R used

Message authentication



Two algorithms:

(1) Tag(K,Msg) outputs a tag T

(2) Verify(K,Msg,T) outputs 0/1 (invalid / valid)

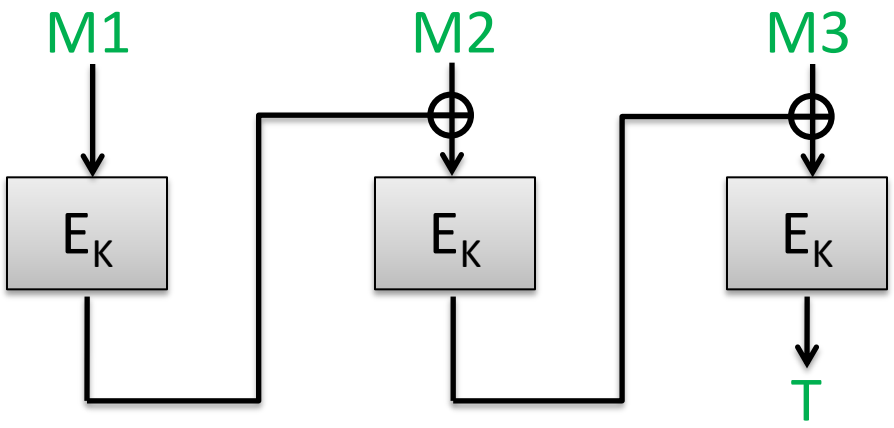
Security: No computationally efficient attacker can forge tags for a new message even when attacker gets

$$(Msg_1 , T_1) , (Msg_2 , T_2), \dots , (Msg_q , T_q)$$

for messages of his choosing and reasonably large q.

CBC-MAC

Split Msg into blocks M1, M2, M3

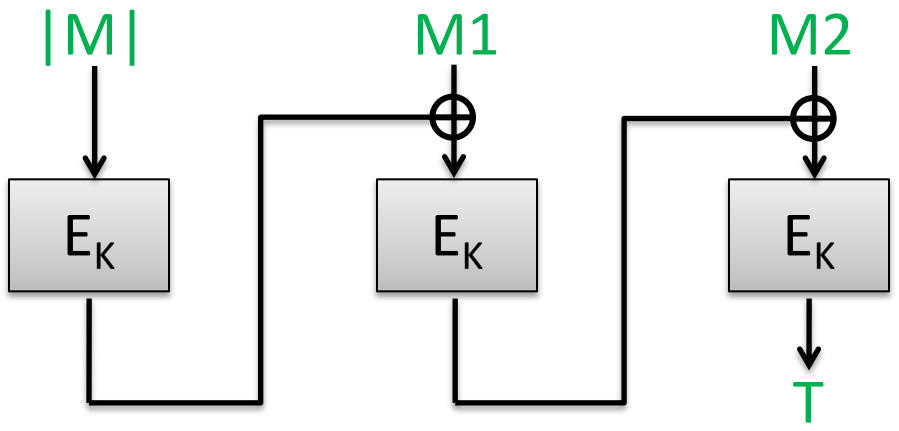


Turns out this is secure MAC

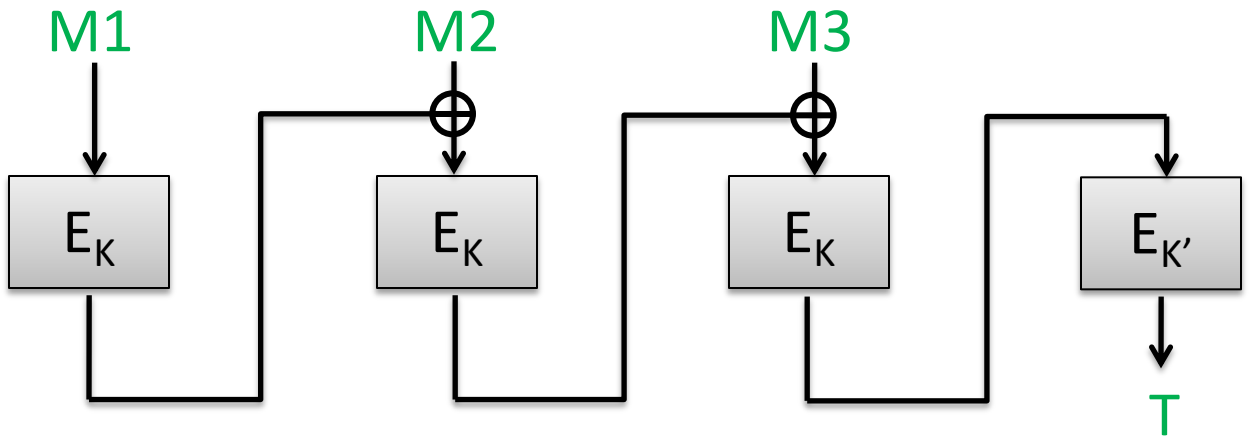
if K used only on same-length messages

Variable-message-length CBC-MAC

- Prepend message length

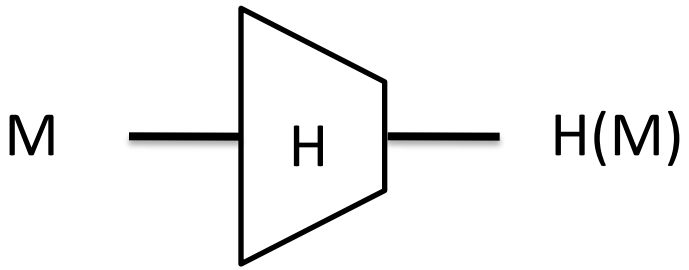


- Encrypted CBC-MAC



Hash functions and message authentication

Hash function H maps arbitrary bit string to fixed length string of size m

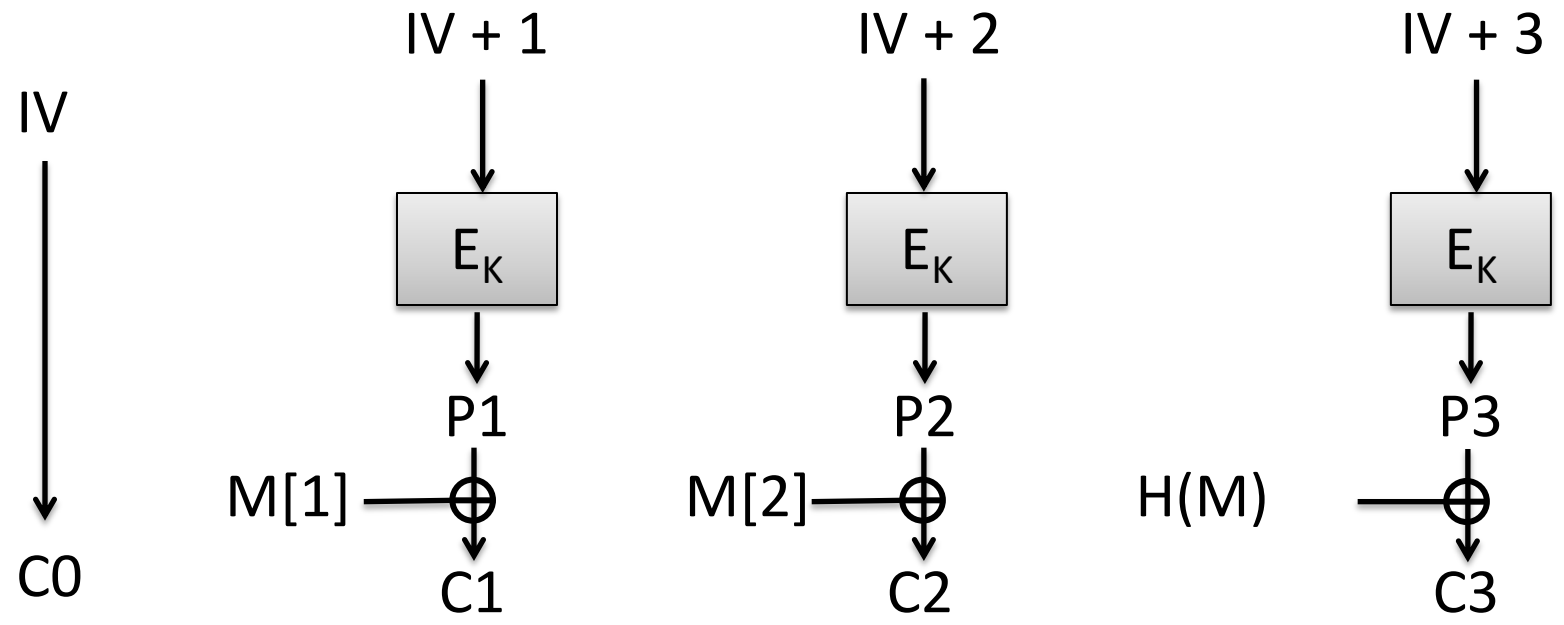


MD5: m = 128 bits
SHA-1: m = 160 bits
SHA-256: m = 256 bits

Some security goals:

- collision resistance: can't find M != M' such that H(M) = H(M')
- "behaves like" random function: H(M) is uniformly distributed bit string for all M
  - Sometimes called the random oracle model (ROM)

Non-cryptographic checksums?

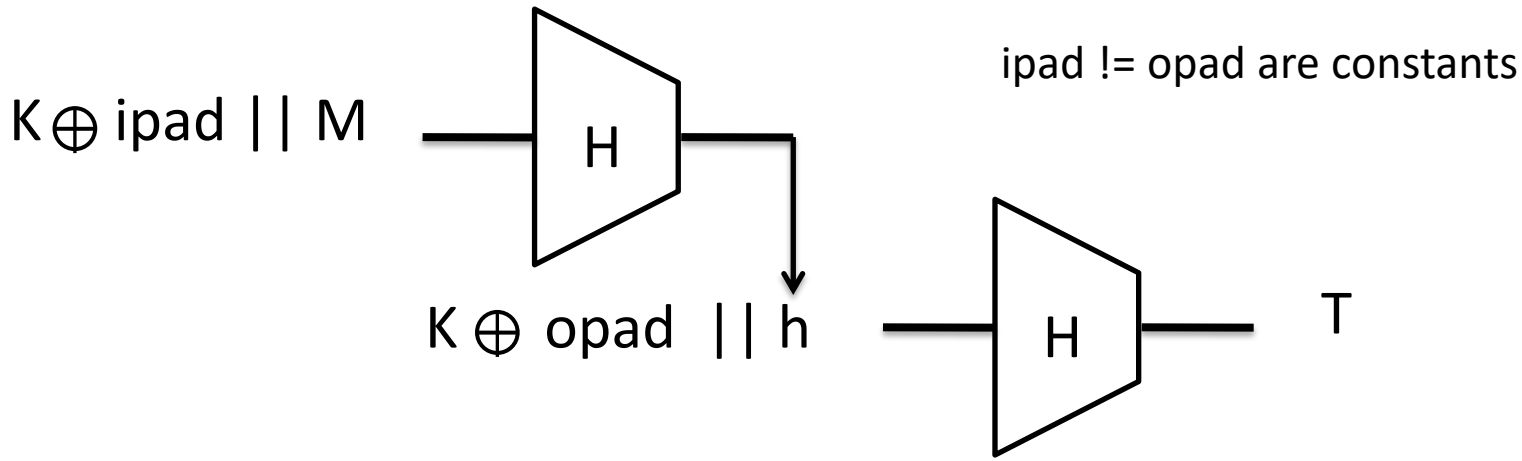


Can simply maul message and H(M) checksum to ensure correctness
Attacks work against ***any*** public computable function

Message authentication with HMAC

Use a hash function H to build MAC
Kg outputs uniform bit string K

Tag(K,M) = HMAC(K,M) defined by:

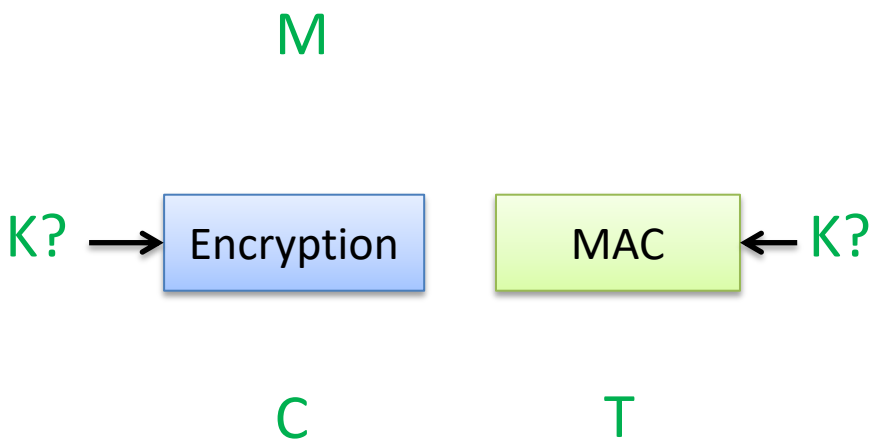


To verify a M,T pair, check if HMAC(K,M) = T

Unforgeability holds if H behaves like a random function

Build *authenticated encryption…?*

- Recall that our goal is authenticated encryption
- Want to combine some encryption scheme with a MAC – how do we do this? Any ideas?



Build a new scheme from CBC and HMAC

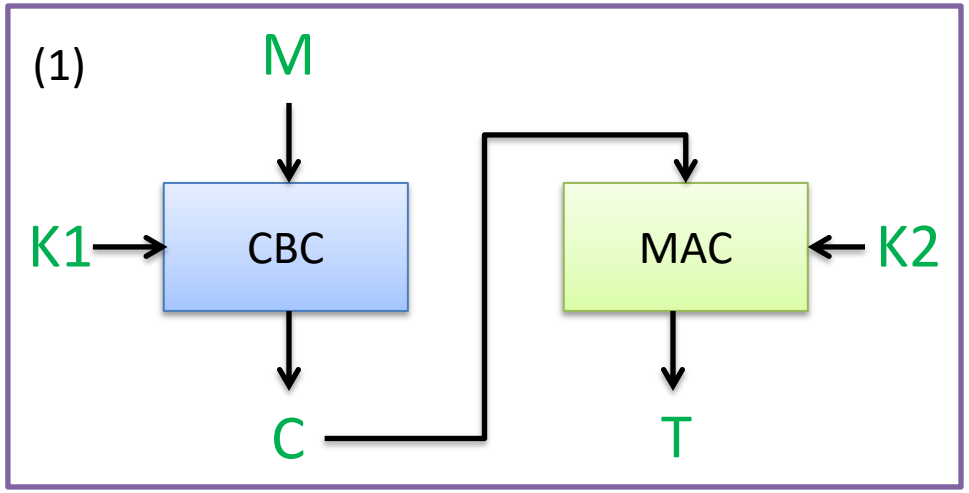
Kg outputs CBC key K1 and HMAC key K2

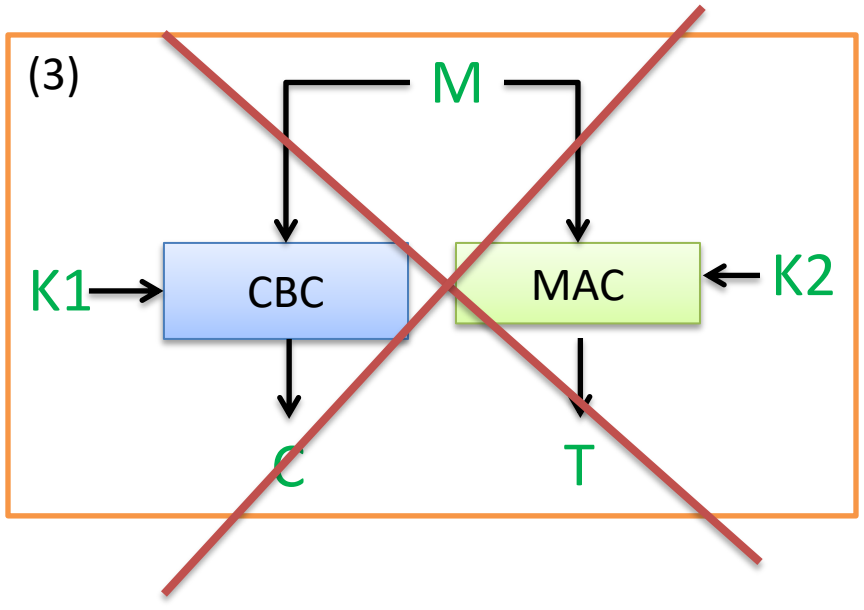
Several ways to combine:

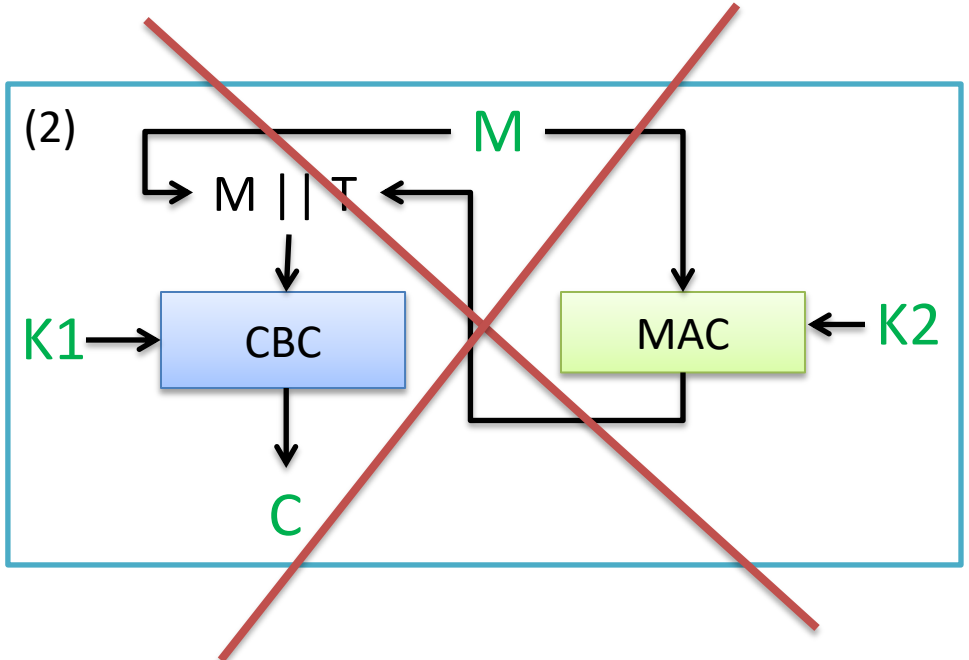
(1) encrypt-then-mac

(2) mac-then-encrypt

(3) encrypt-and-mac

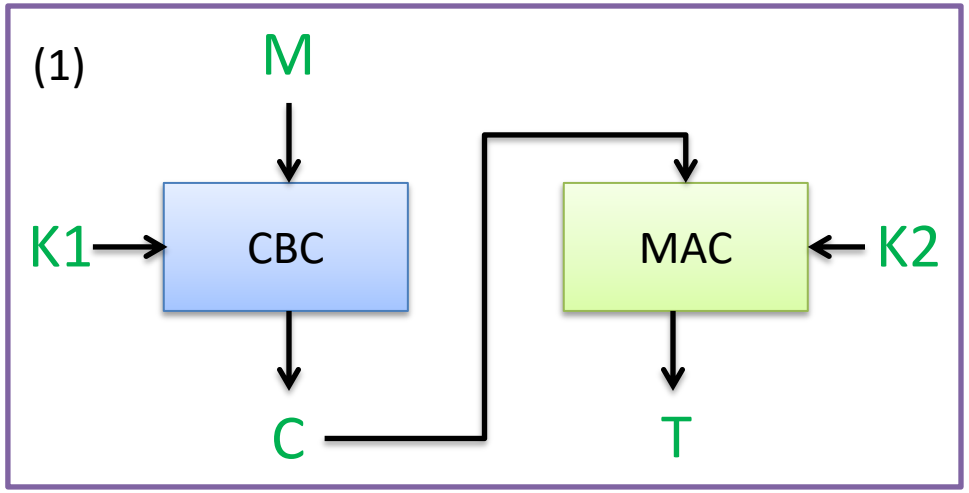






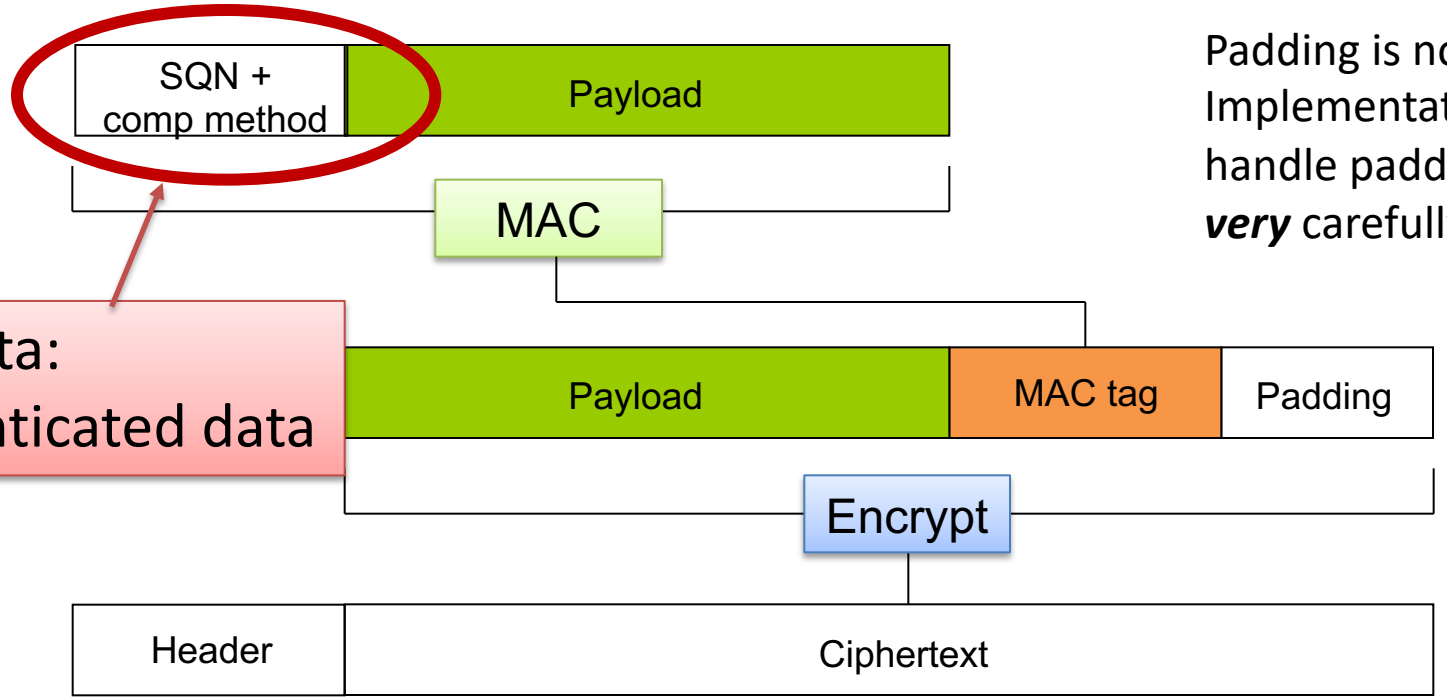
Build a new scheme from CBC and HMAC
Kg outputs CBC key K1 and HMAC key K2

Several ways to combine:
(1) encrypt-then-mac
(2) mac-then-encrypt
(3) encrypt-and-mac



Thm. If encryption scheme provides confidentiality against passive attackers and MAC provides unforgeability, then Encrypt-then-MAC provides secure authenticated encryption

TLS 1.2 record protocol: MAC-Encode-Encrypt (MEE)



Associated data:
public, authenticated data

Padding is not MAC'd. Implementations must handle padding checks ***very*** carefully.

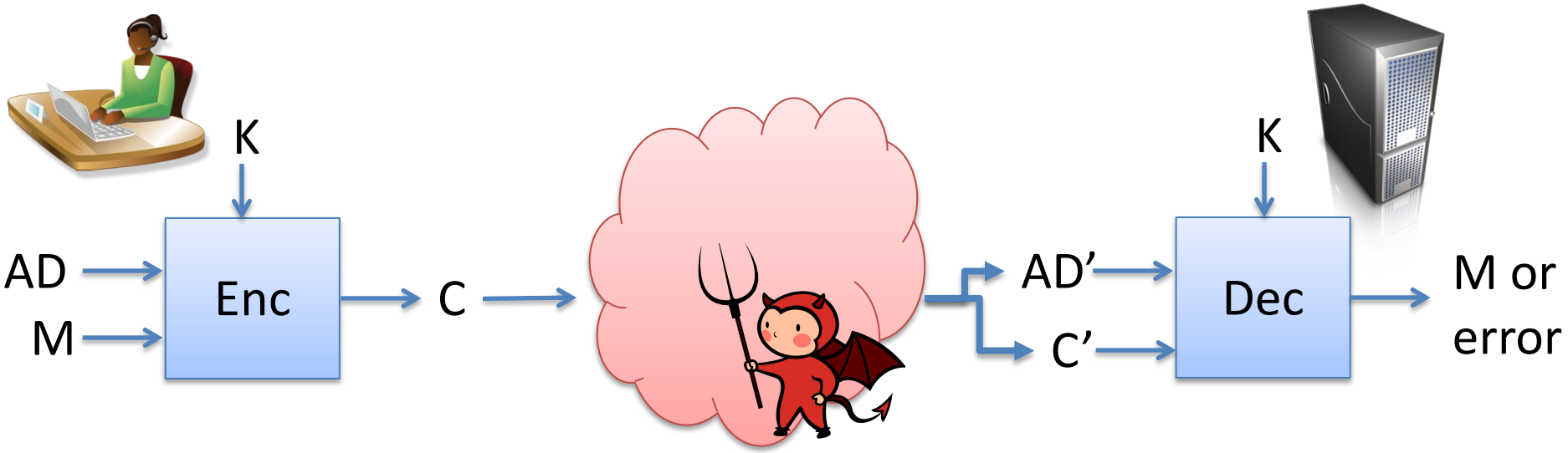
Padding slightly different than PKCS#7 (subtract 1 from each byte):
00
01 01
02 02 02
…

MAC — HMAC-MD5, HMAC-SHA1, HMAC-SHA256

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

Now TLS uses better authenticated-encryption schemes, this is deprecated

Authenticated encryption w/ associated data (AEAD)



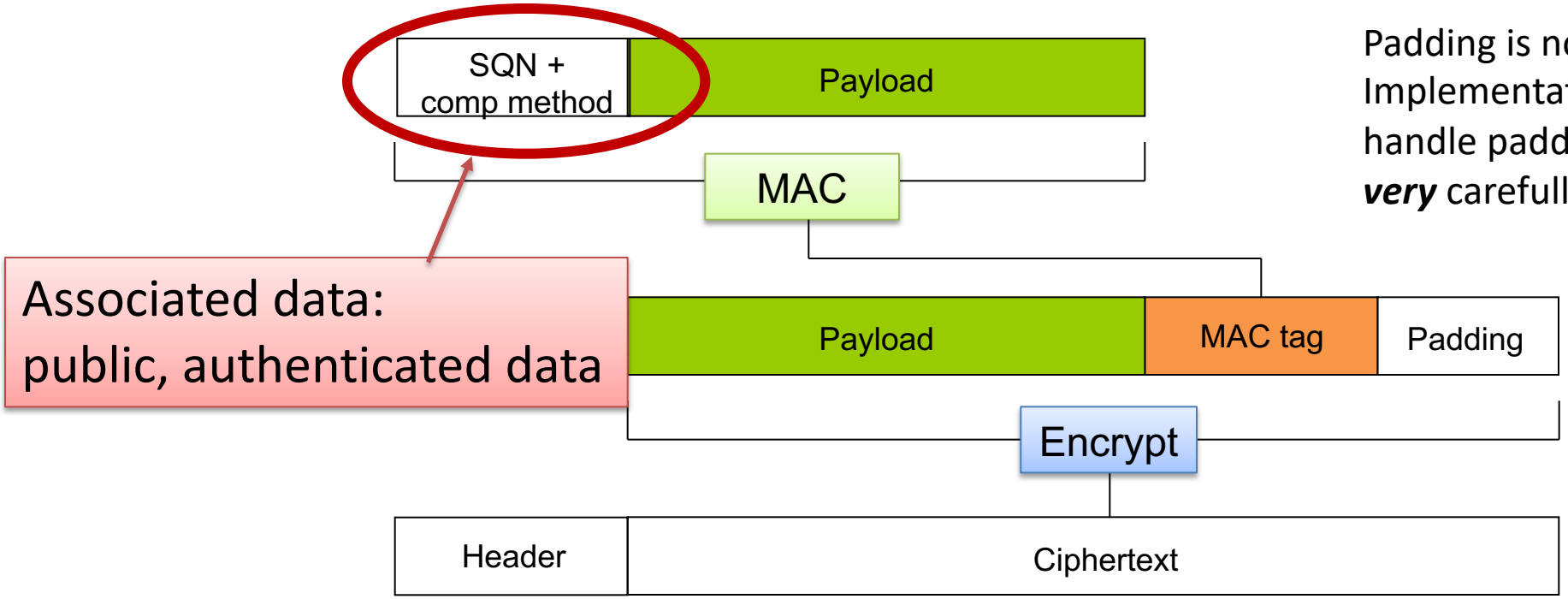
Associated data ***need not be confidential***, but ***must be authenticated***

Typically used for cleartext headers sent

- TLS 1.2 example: sequence number, compression methods
- Context information (database row number, disk sector number, etc.)

All deployed AE schemes support AD now

TLS 1.2 record protocol: MAC-Encode-Encrypt (MEE)



Associated data:
public, authenticated data

Padding is not MAC'd. Implementations must handle padding checks ***very*** carefully.

Are implementions of this MEE construction likely vulnerable to padding oracle attacks?

CCAs and deployed schemes

Attack	Description	Year
Vaudenay	10's of chosen ciphertexts, recovers message bits from a ciphertext. Called "padding oracle attack"	2001
Canvel et al.	Shows how to use Vaudenay's ideas against TLS	2003
Degabriele, Paterson	Breaks IPsec encryption-only mode	2006
Albrecht et al.	Plaintext recovery against SSH	2009
Duong, Rizzo	Breaking ASP.net encryption	2011
Jager, Somorovsky	XML encryption standard	2011
Duong, Rizzo	"Beast" attacks against TLS	2011
AlFardan, Paterson	Attack against DTLS	2012
AlFardan, Paterson	Lucky 13 attack against DTLS and TLS	2013
Albrecht, Paterson	Lucky microseconds against Amazon's s2n library	2016

Dedicated authenticated encryption schemes

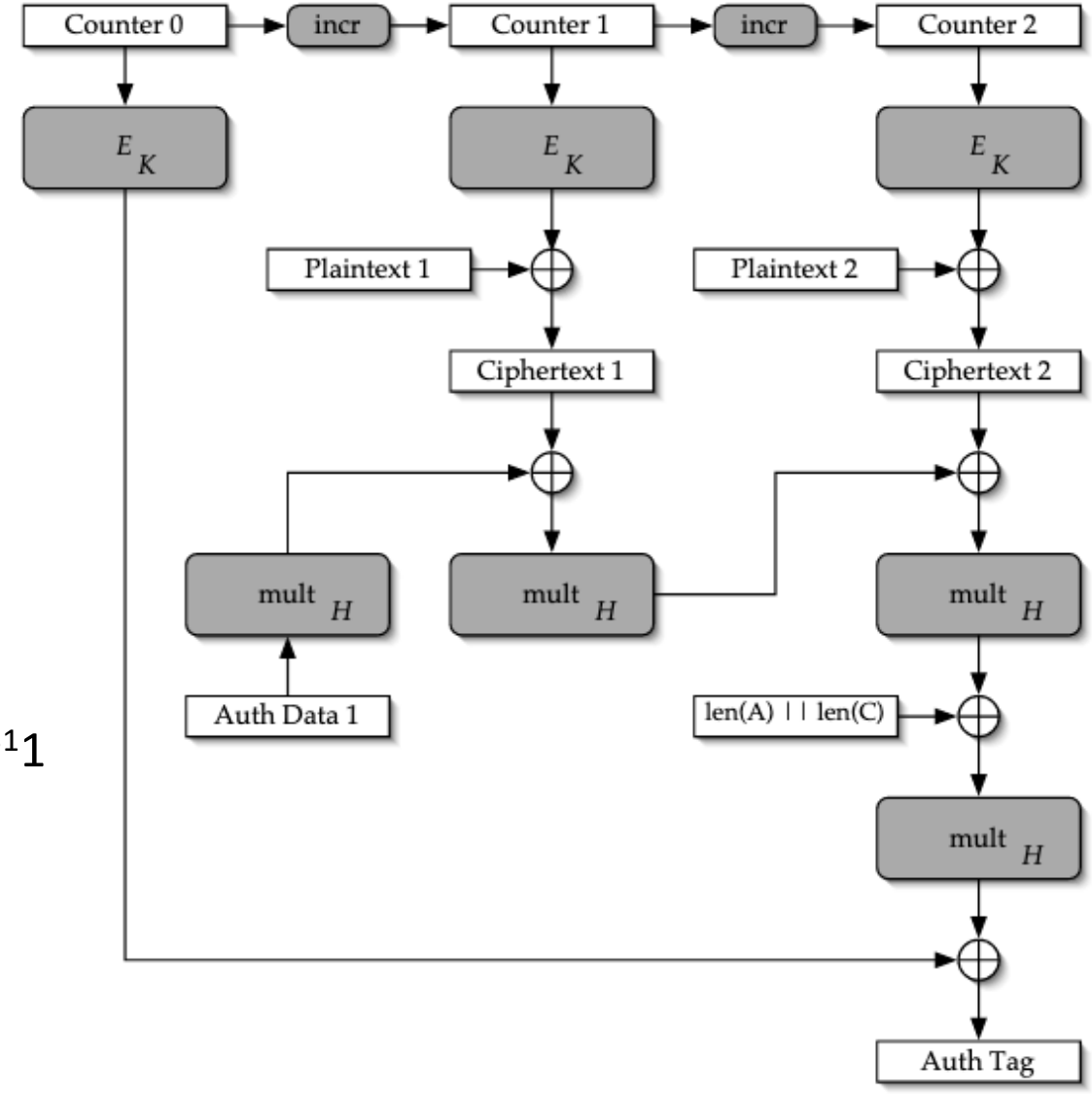
Not a generic composition of Enc, MAC.
Directly construct from blockcipher, stream cipher, etc.

Attack	Inventors	Notes
OCB (Offset Codebook)	Rogaway	One-pass (one blockcipher call per block of message)
GCM (Galois Counter Mode)	McGrew, Viega	CTR mode plus specialized MAC
Salsa20/Poly1305	Bernstein	Stream cipher plus Carter-Wegman MAC
CWC	Kohno, Viega, Whiting	CTR mode plus Carter-Wegman MAC
CCM	Housley, Ferguson, Whiting	CTR mode plus CBC-MAC
EAX	Wagner, Bellare, Rogaway	CTR mode plus OMAC (variant of CBC-MAC)

Crypto community in process of designing new ones due to commitment attacks & scaling issues

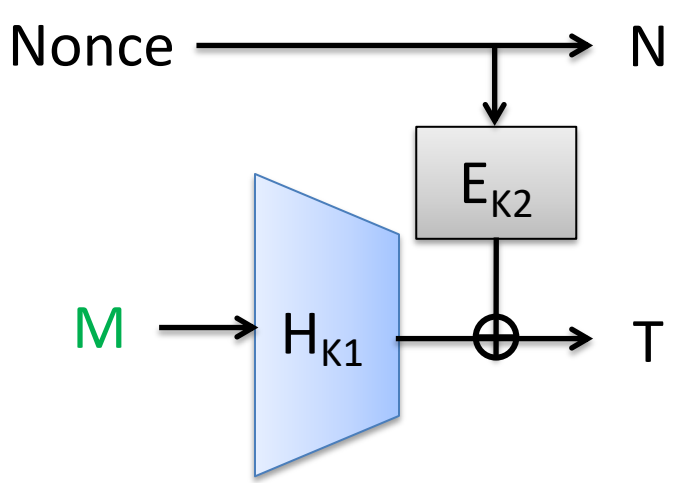
AES Galois Counter Mode

- Counter-mode encryption
- Carter-Wegman style MAC that uses Galois field (aka finite field) multiplications to define universal hash



$H <- E_K(IV||0^{32})$
Counter = IV || $0^{31}1$

Carter-Wegman message authentication schemes



Nonce must be fresh for each M
Random value or counter
If repeated, security fails

Intuition:
Hide universal hash by encrypting it

A keyed function $H : \{0,1\}^k \times \mathcal{M} \rightarrow \{0,1\}^n$ is an ϵ-almost universal (AU) hash function if for all $M \neq M'$

$$\Pr\left[\, H_K(M) = H_K(M') \,\right] \leq \epsilon$$

where the probability is over choice of K.

Symmetric Encryption Advice

Never use CTR mode or CBC mode by themselves

Passive security is almost never good enough!!

Encrypt-then-MAC best way to do generic composition

Used in Signal, for example

Unfortunately no widely supported standard (yet)

Dedicated modes that have been analyzed thoroughly are way to go in practice

More topics in symmetric encryption

- Misuse-resistant authenticated encryption
  - Better security should IV/nonce accidentally repeats
- Format-preserving encryption
  - Encrypt credit card numbers so that ciphertexts are syntactically valid credit cards
  - Variant is format-transforming, used previously in censorship-resistance systems
- Committing authenticated encryption
  - Vulnerabilities in Facebook abuse reporting due to use of AES-GCM
  - Encrypt-then-HMAC provides solution

Dedicated authenticated encryption schemes

Attack	Inventor(s)	Notes
OCB (Offset Codebook)	Rogaway	One-pass mode and fastest
AES-GCM (Galois Counter Mode)	McGrew, Viega	CTR mode plus Carter-Wegman MAC
ChaCha20/Poly1305	Bernstein	"essentially" CTR mode plus special Carter-Wegman MAC
CCM	Housley, Ferguson, Whiting	CTR mode plus CBC-MAC
EAX	Wagner, Bellare, Rogaway	CTR mode plus OMAC

Other considerations in AE:

robustness & IV misuse, deterministic AE, associated data, …

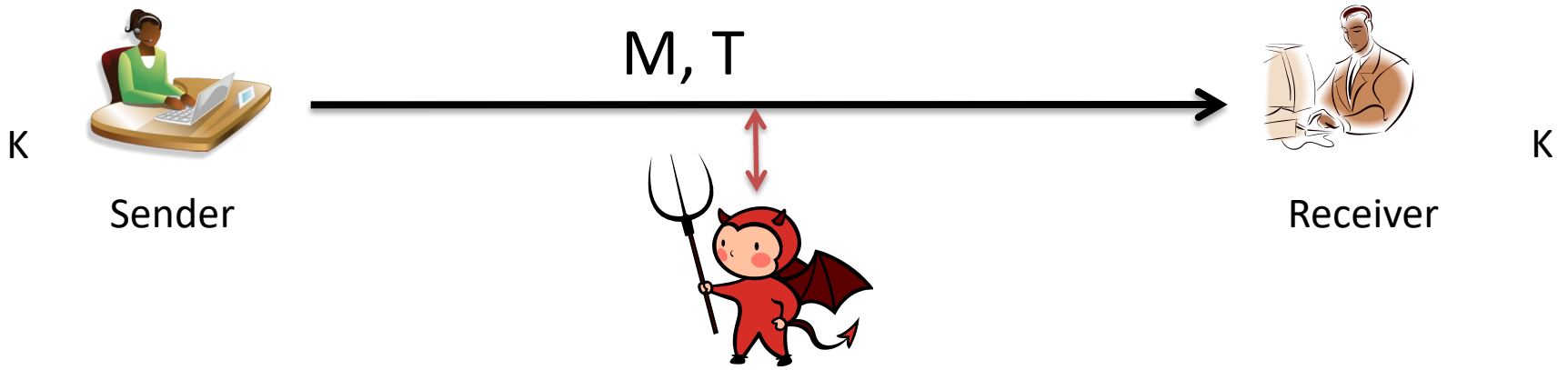
key and context commitment security (i.e., Julia's presentation)

Next time: asymmetric cryptography

So far we've seen how to encrypt using shared key

We'll introduce asymmetric cryptography next lecture

Message authentication schemes



Scheme is triple of algorithms MA = (Kg,Tag,Verify):

(1) Kg outputs secret key K

(2) Tag(K,M) outputs a tag T

(3) Verify(K,M,T) outputs 0/1 (invalid / valid)

Message authentication code (MAC):
Tag deterministic +
Verify checks by recomputing Tag

Correctness: Verify(K,M,Tag(K,M)) = 1 always

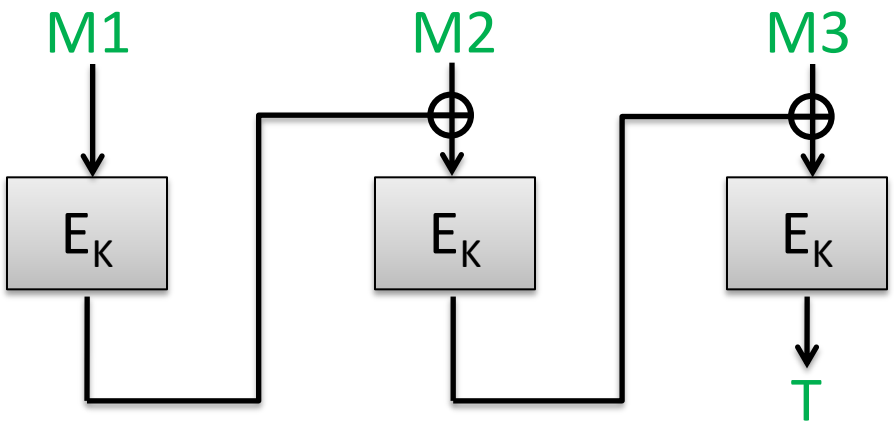
Security: No computationally efficient attacker can forge tags for a new message even when attacker gets

$$(M_1 , T_1) , (M_2 , T_2), \dots , (M_q , T_q)$$

for messages of their choosing and reasonably large q.

CBC-MAC

Message authentication code (MAC)



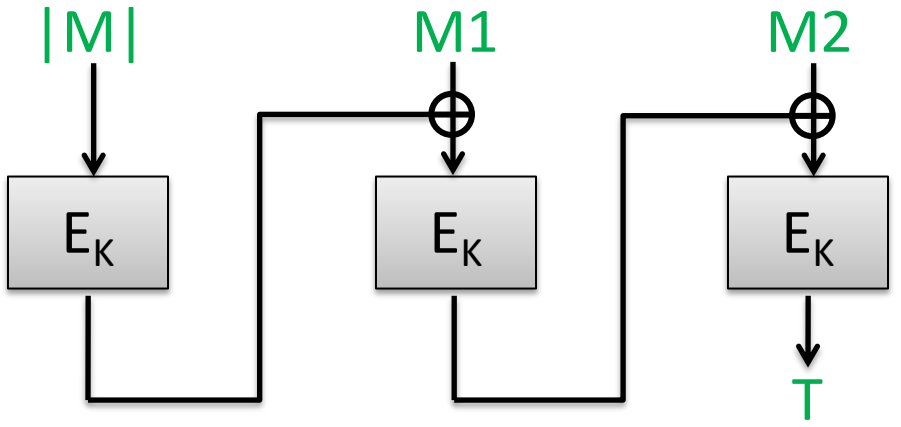
Turns out this is (provably) a good PRF

if K used only on same-length messages

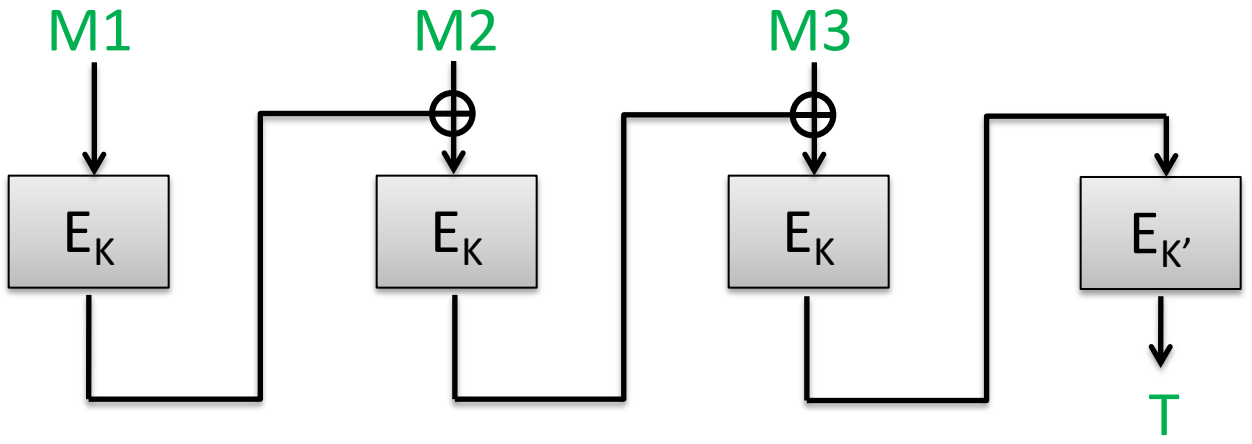
http://blog.cryptographyengineering.com/2013/02/why-i-hate-cbc-mac.html

Secure variable-message-length CBC-MAC

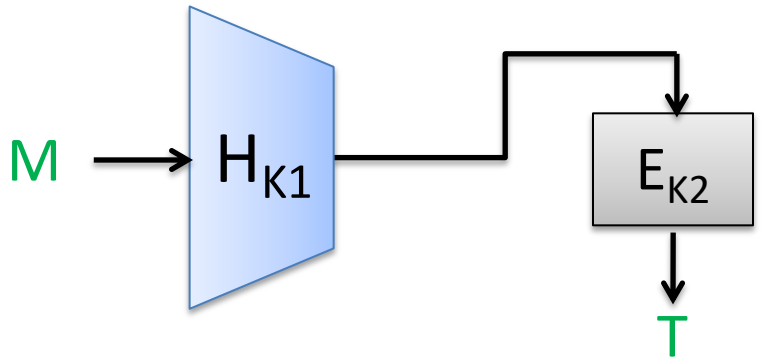
- Prepend message length



- Encrypted CBC-MAC



Universal hash then PRF approach



A keyed function $H : \{0,1\}^k \times \mathcal{M} \rightarrow \{0,1\}^n$ is an ϵ-almost universal (AU) hash function if for all $M \neq M'$

$$\Pr\left[\, H_K(M) = H_K(M') \,\right] \leq \epsilon$$

where the probability is over choice of K.

Turns out this is a good construction for any H that is a (computational) ***universal hash function***

Intuition: E_{K2} hides info about K1 unless collision. For q queries, chance of collision at most

$q^2 \epsilon / 2$

Symmetric Encryption Advice

Never use CTR mode or CBC mode by themselves

Passive security is almost never good enough!!

Encrypt-then-MAC best way to do generic composition

Used in Signal, for example

Unfortunately no widely supported standard (yet)

Dedicated modes that have been analyzed thoroughly are way to go in practice