Authenticated encryption

Recap: Collision resistant hash $H: M \to T$

hard to find $m_0 \neq m_1$ s.t. $H(m_0) = H(m_1)$

construction: SHA2: Davies-Meyer compr. func. $\Rightarrow$ Merkle-Damgard

most widely used in practice

SHA256, SHA384, SHA512

SHA3: Sponge.

birthday attack $\Rightarrow$ output ≥ 256 bits

Applications:

(1) MAC for 256-bit msgs $\Rightarrow$ Big MAC (not used)

(2) Software package integrity: Files $F_1, \ldots, F_n$.

need public read-only space to store hashes.

attacker cannot find F' s.t. $H(F_5) = H(F')$

$\Rightarrow$ if downloaded file F_5 has correct hash then file is authentic.

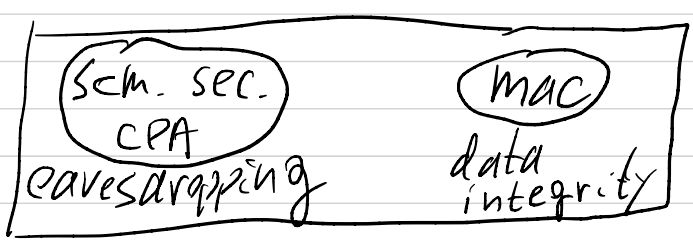
MACs: Secure PRF $\Rightarrow$ secure MAC.

Big PRF from small PRF: (1) CBC-MAC or PMAC

(2) HMAC: black box use of SHA256

(3) Carter-Wegman MAC

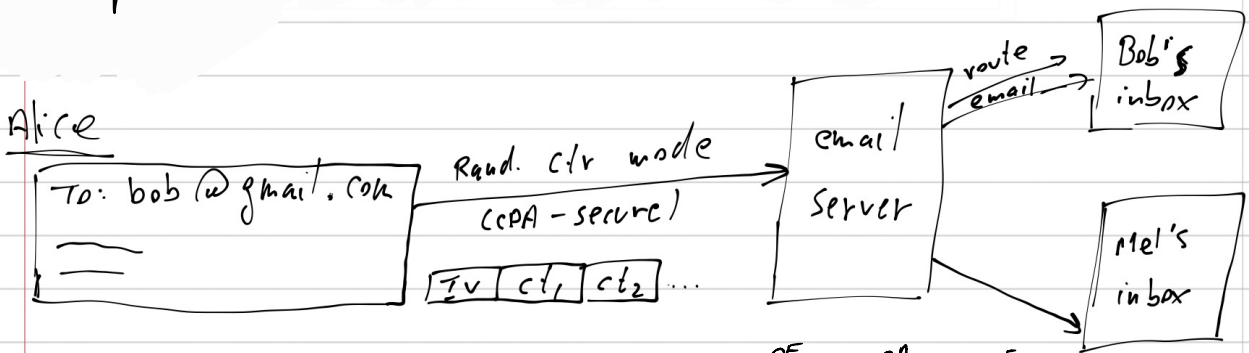
So far:



Authenticated encryption: combining conf. + integrity

central point: CPA security w/o integrity provides no confidentiality!!

Example:



Attacker Mel: $\hat{ct}_1 \leftarrow ct \oplus [000 \| \overbrace{'b' \oplus 'm'}^{0F} \| \overbrace{'o' \oplus 'e'}^{0A} \| \overbrace{'b' \oplus 'l'}^{0E} \| 0 \ldots 0]$

send IV | $\hat{ct}_1$ | ct2 | ...

upon decryption, server gets To: mel@gmail.com.

Lesson: must ensure CT integrity.

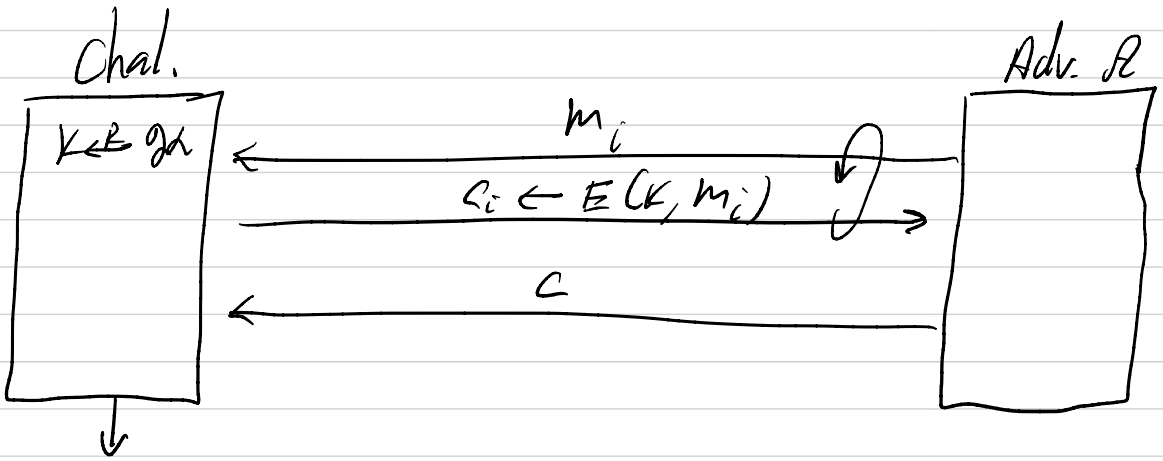
otherwise no confidentiality.

Auth enc: Goal: CPA security + ciphertext integrity

First: if (E, D) is to provide A.E. then

alg $D(k, c)$ outputs msg m,
or a special message "reject" that
indicates that decryption failed.

ciphertext integrity game:



$\mathcal{A}$ wins game if $D(k, c) \neq$ "reject" and $c \notin \{c_1, c_2, \ldots\}$.

Def: (E, D) has C.I. if $\forall$ "eff" $\mathcal{A}$ prob. $\mathcal{A}$
wins the C.I. game is "negl."

Example: rand-ctr-mode is CPA-secure, but no C.I.
adv. requests $c \leftarrow E(k, m)$ for some m, outputs $c \oplus 1^n$.

Def: (E, D) provides **auth. enc.** if

(E, D) is both CPA-secure and has C.I.

$\Rightarrow$ The only form of enc. to use in practice!!

Constructions using MAC & cipher

$\Pi = (S, V)$ secure MAC, (E, D) - CPA secure cipher

$K = (K_e, K_m)$

options:

1. MAC-then-enc (TLS 1.0): $t \leftarrow S(K_m, m)$, $c \leftarrow E(K_e, m \| t)$, send c
2. enc-then-mac (GCM): $c \leftarrow E(K_e, m)$, $t \leftarrow S(K_m, c)$, send (c, t)
3. enc-and-mac (SSH): $t \leftarrow S(K_m, m)$, $c \leftarrow E(K_e, m)$, send (c, t)

method 3: insecure if MAC leaks info. about m. Not CPA

method 1: can be insecure (POODLE attack). No C.I. on SSL 3.0

Thm: enc-then-mac provides A.E.
whenever (E, D) is CPA secure, and (S, V) is secure MAC.

note: K_e, K_m must be indep. keys. e.g.

$K_e \leftarrow PRF(K, \text{"enc"})$, $K_m \leftarrow PRF(K, \text{"mac"})$

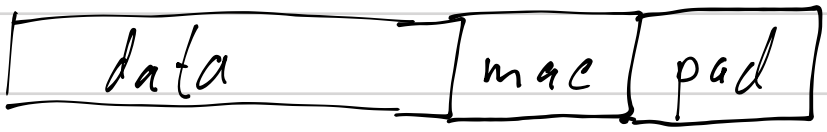
GCM (Galois counter mode):

(nonce-based - ctr - mode enc.) - then - MAC.

Required in TLS 1.3.

The problem with mac-then-enc: (POODLE)

SSL 3.0: CBC-enc.



$\rightarrow ct$

not covered by mac!!

Say $pad = \underbrace{[15|15|\cdots|15]}_{16 \text{ byte dummy block}}$

SSL3.0: pad removed during dec. w/o checking pad structure.

$ct = [c_0 | c_1 | c_2 | c_3]$

attack $ct^* = [c_0 | c_1 | c_2 | c_0]$

if dec. of c_0 ends in 15 then ct^* is valid $ct \Rightarrow$ breaks C.I.

$$Pr[\text{win}] = \frac{1}{256}$$

$\Rightarrow$ adv. learns something about dec. of c_0

$\Rightarrow$ POODLE makes this a real attack to extract secret cookies.

Auth. Enc. w/ Assoc. Data (AEAD)

openssl GCM API: (enc-then-mac)

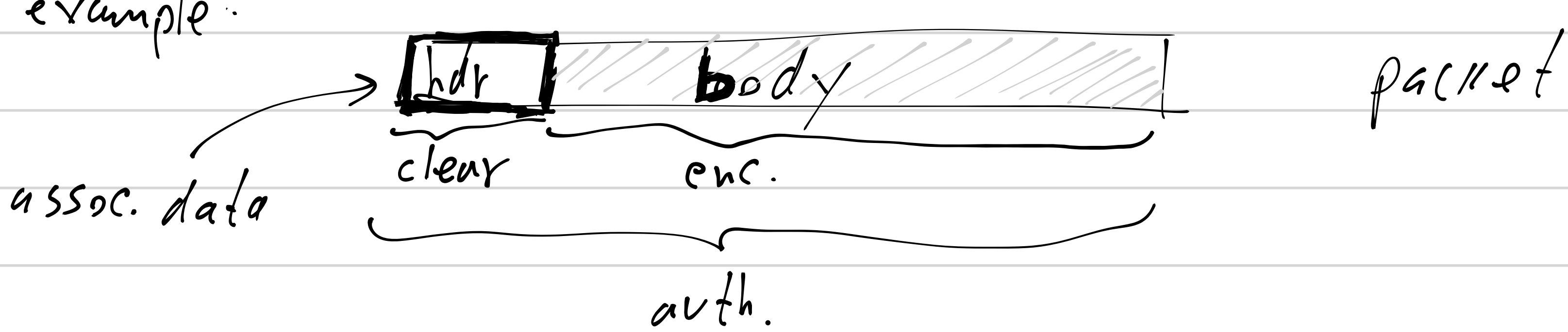
```
int encrypt(
                  char *pt,   int pt_len,     // PT
needed for      { char *add,  int aad_len,    // assoc. data
dec.            { char * key,
                { char * iv,                  // nonce
CPA enc.        { char * ct                   // output
mac             { char * tag   )
```

aad: additional assoc. data.

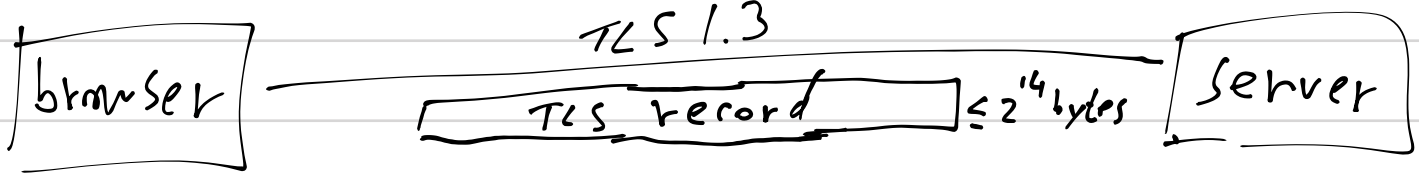
$c \leftarrow E(K_e, m), \quad t \leftarrow S(K_m, \text{aad} \| c), \quad \text{send } (c, t)$

aad - not part of ct.

example:



Case study: GCM in TLS 1.3:



Unidirectional keys: $(k_{B\to S}, k_{S\to B}) \leftarrow$ HKDF(master-secret + random)

Stateful encryption: each side maintains 64-bit counter

wsc_B, wsc_S – write_seq_ctr

init to 0 at session setup.

wsc_B ++ when browser sends record. server does same when recieves record.

- When browser sends to server:

also known to server:

key = $k_{B\to S}$ = (enc key, mac key)

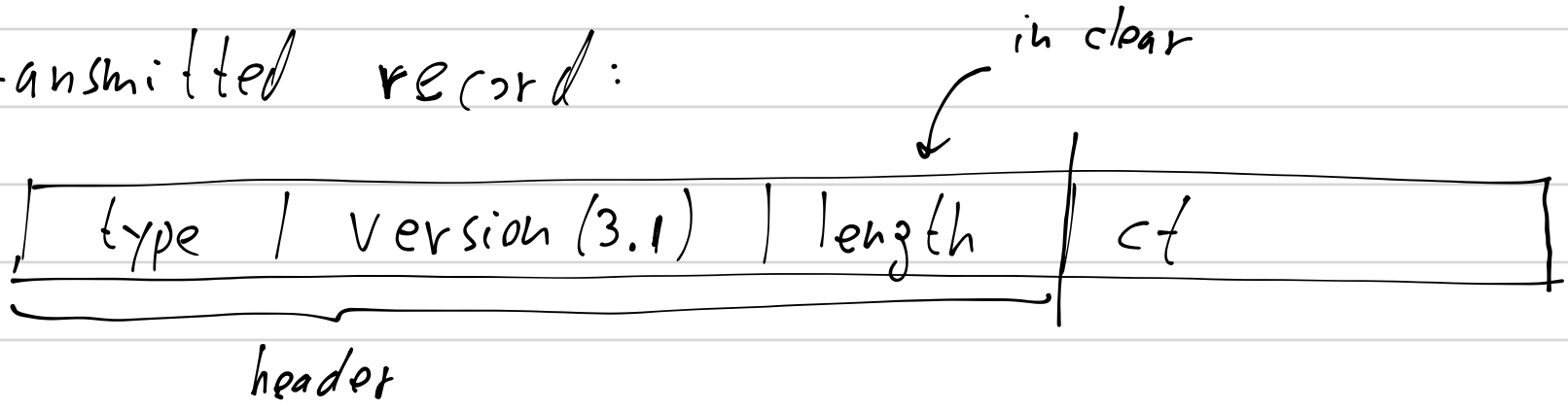
assoc. data = (wsc_B, record-type, prot. version "3.1")

nonce = $wsc_B \oplus$ (client_write_iv)

generated at session setup and then fixed

wsc – prevents replay, re-ordering, and deletion.

Transmitted record:



note: assoc. data and nonce are not sent over the network