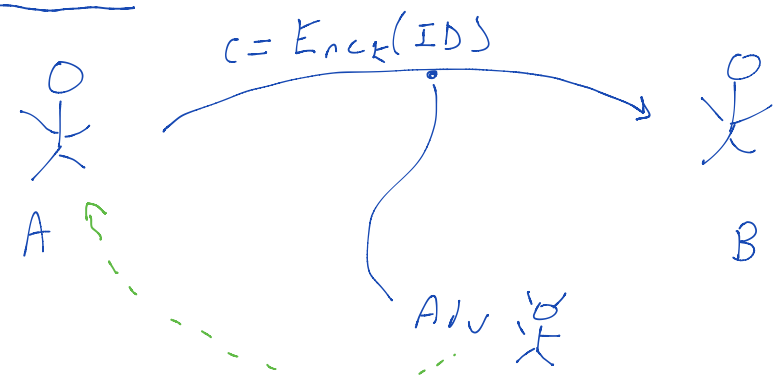
Lecture 6

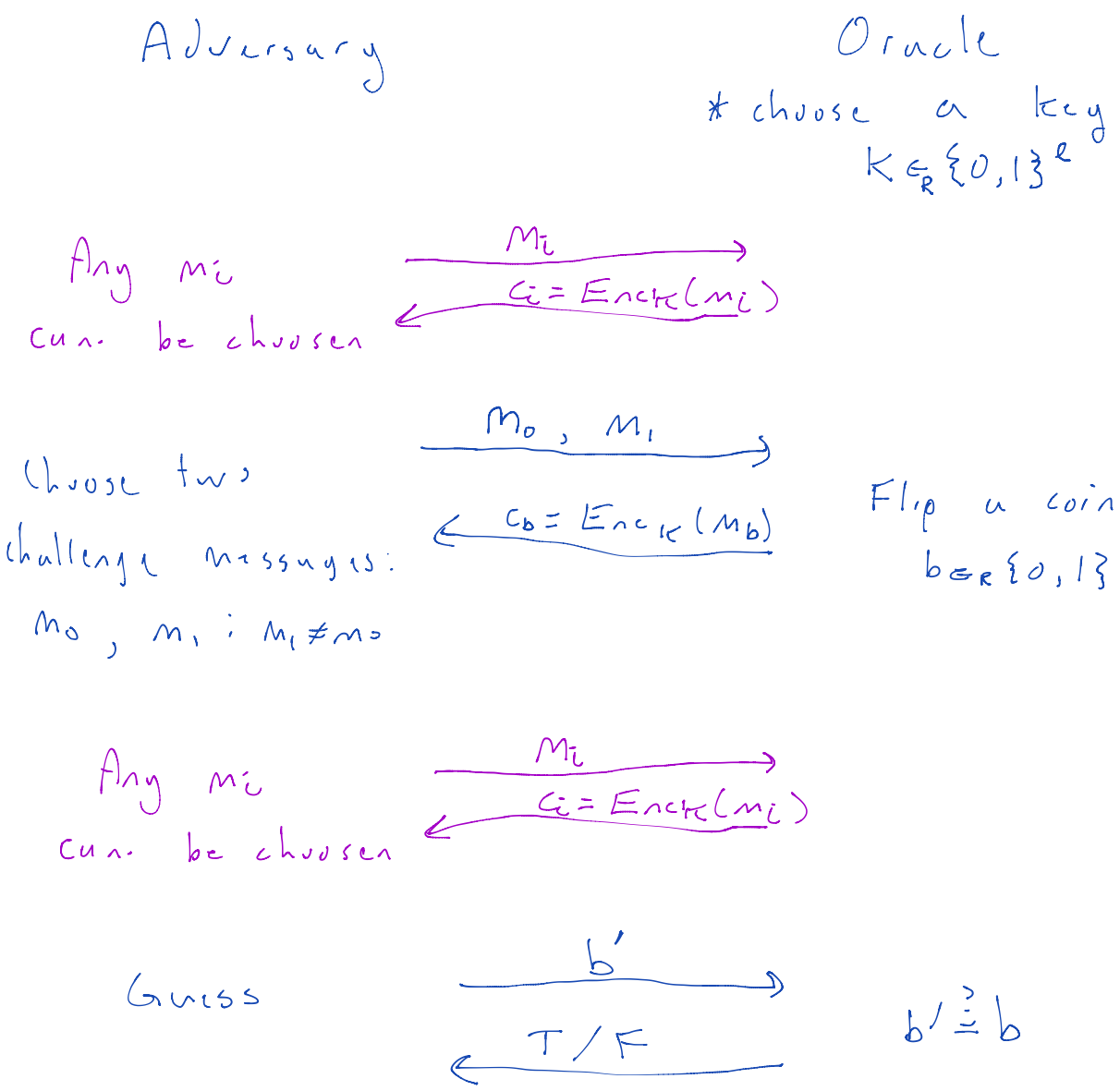


} CPA-security

Randomization is necessary.

	Requirement	Examples
OTS		ECB
CPA	Randomized	CBC, CTR, Stream Cipher
CCA	Non-malleability	CPA-secure Enc + MAC, GCM

CPA Game



Win → Not CPA secure

↳ Winning percentage $> \frac{1}{2} + \varepsilon$

CCA Game

Adversary

Oracle

* choose a key $K \in_R \{0,1\}^\ell$

Any m_i can be chosen

m_i →

← $c_i = Enc_K(m_i)$

Any c_i can be chosen

c_i →

← $m_i = Dec_K(c_i)$

Choose two challenge messages: $m_0, m_1 : m_1 \neq m_0$

m_0, m_1 →

← $c_b = Enc_K(m_b)$

Flip a coin $b \in_R \{0,1\}$

Any m_i can be chosen

m_i →

← $c_i = Enc_K(m_i)$

Any $c_i \neq c_b$ can be chosen

c_i →

← $m_i = Dec_K(c_i)$

Guess

b' →

← T / F

$b' \stackrel{?}{=} b$

Win → Not CCA secure

↳ Winning percentage $> \frac{1}{2} + \varepsilon$

Fact: CBC is not CCA-secure:

↳ m_0 = [000...0] [000...0]

m_1 = [111...1] [111...1]

Assume oracle picks m_b

c_b = [ ] [ ]

↳ can't ask for decryption of

↳ c_b' = [flip] [ ]

↳ ask for decryption of c_b'

↓

m_b' = [ ] [111011111...1]

Guess → b=1 → win 100%

Fact: CTR mode is not CCA-secure

↳ m_0 = [0000...0]

m_1 = [1111...1]

c_b = [ ]

c_b' = [ ]

M_b' = { 000100000... → b=0 ; 1111011111... → b=1 } win 100%

Fact: no malleable encryption scheme is CCA-secure

↳ always modify c_b to create c_b' and ask for decryption and use the malleability to distinguish m_0 & m_1.

Block Ciphers

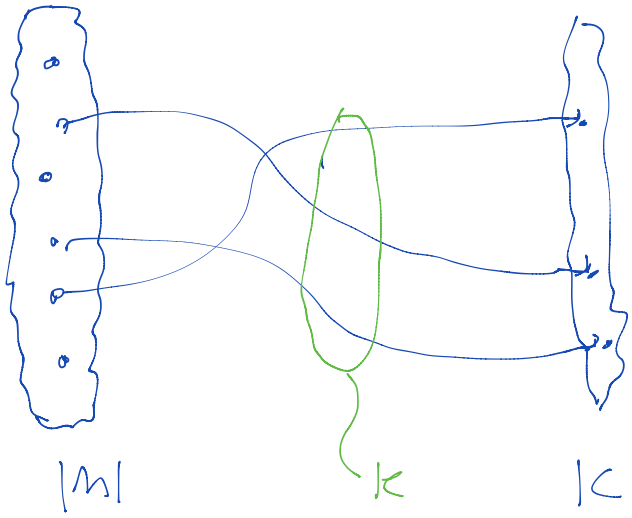
	Key	
DES	56/40	✗
3DES	112	~
AES	128/192/256	✓

AES ← winning bid of a competition run by NIST

↳ Confusion → S-box → maps values in a non-linear way.

↳ Diffusion → spreading values around → ShiftRow, Mix column

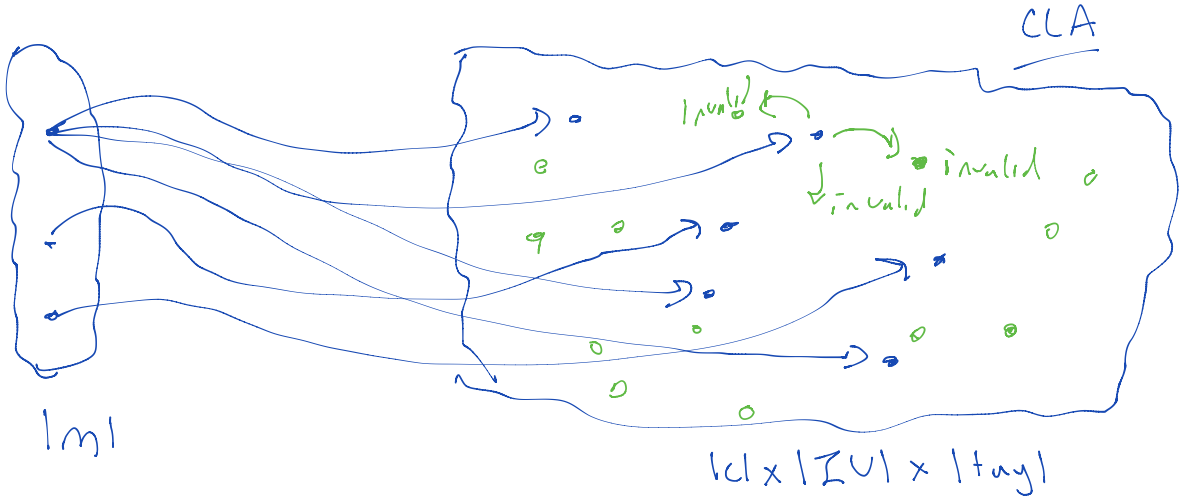
↳ Incorporate the key (secret)



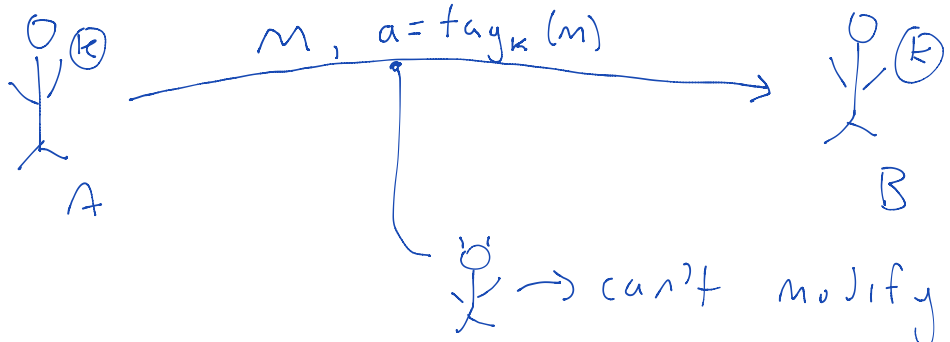
OTS

$|M| = |C|$

CPA



Message Authentication Codes (MAC)



→ can't modify m and generate the correct tag w/o knowing K

↳ if Eve changes $m \to m'$, cannot compute $a' = tag_K(m')$

MAC

$$a = tag_K(m)$$

$m \in \{0,1\}^*$

$K \in \{0,1\}^{128}$ ← at least

a: fixed length: $\{0,1\}^d$

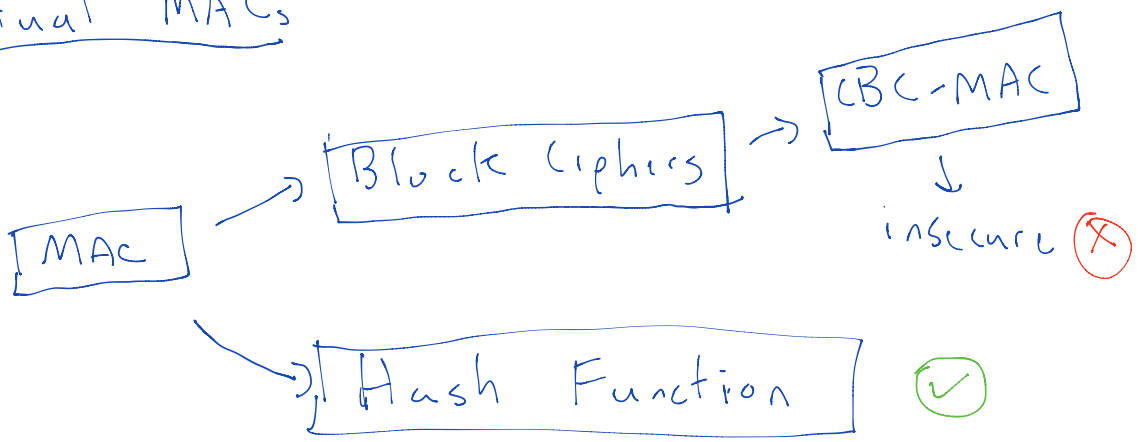
Alice sends $\langle m, a \rangle$

↳ no encryption!

Bob: $T/F = Verify_k(m, a)$

$= \{ MAC_k(m) \stackrel{?}{=} a \}$

Actual MACs



Security

* Infeasible to create a valid tag $a = MAC_K(m)$ on chosen message m w/o knowing the key.
* Best attack should be exhaustive search on the key ($W \approx 2^{112}$).

Not security properties:

* Doesn't matter if adversary can recover m from a

Hash-based MACs

Attempt 1 (X):

$$a = H(m)$$

↳ known function

Attempt 2 (X):

$$a = H(k \| m)$$

k: key; m: message; a: fixed length

Forgery using LEA

see $\langle m, a \rangle$

↳ $\langle m' = m \| z,\ a' = H(k \| m \| z) \rangle$

LEA:

LEA — don't know

See: $H((k \| m))$

Compute: $H((k \| m) \| z)$

don't know — choose

$H(k \| m \| z)$: don't know.

Attempt 3 (X):

$a = H(m||k)$

↳ LEA: $H(m||k||z)$

↓

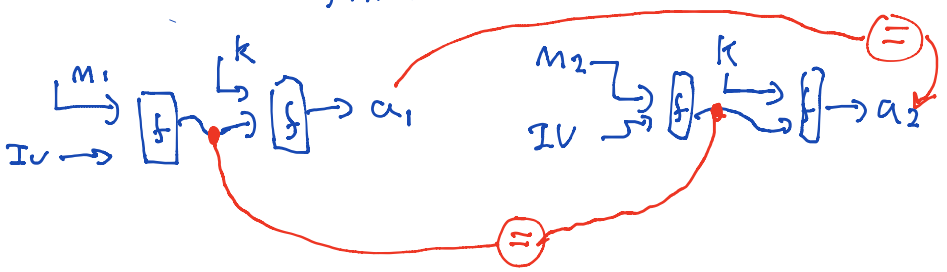
$m||k, z$ ✗ attack doesn't work.

↑ key.

Consider C.R. of Hash function

↳ say you have $H(m_1) = H(m_2)$

↳ then MACs will be same.



↳ if MACs are same, it is a forgery. (w/o knowing key)

↳ Requires H to be CR

which requires output

of H to be ≥ 224 bits.

↳ Nice to use a H that

only has to be PR

Attempt 4 (✓):

HMAC

↳ $a = \underbrace{H(k \| \overbrace{H(k \| m)}^{\text{inner}})}_{\text{outer.}}$

↳ simplified

↳ real one has padding (see textbook)

Inner: $z = H_1(k \| m)$

Outer: $a = H_2(k \| z)$ → LEA

LEA → attempt 2

$H_1(k||m)$ ← don't know value
↳ b/c H_2 is PR
↳ can't do LEA

$H_2(k||z||m)$
doesn't change message
↳ go inside H_1

HMAC is a secure MAC that only requires hash to be PR.

Implementation Note:

MAC values are often truncated to small values (e.g., 64-bits) because they only need to be secure for lifetime of a conversation

↳ if someone can break a MAC in 10 years, it isn't that useful

(as opposed to encryptions).