Lecture 3: Block ciphers

Recap: OTP has perfect secrecy, but long keys

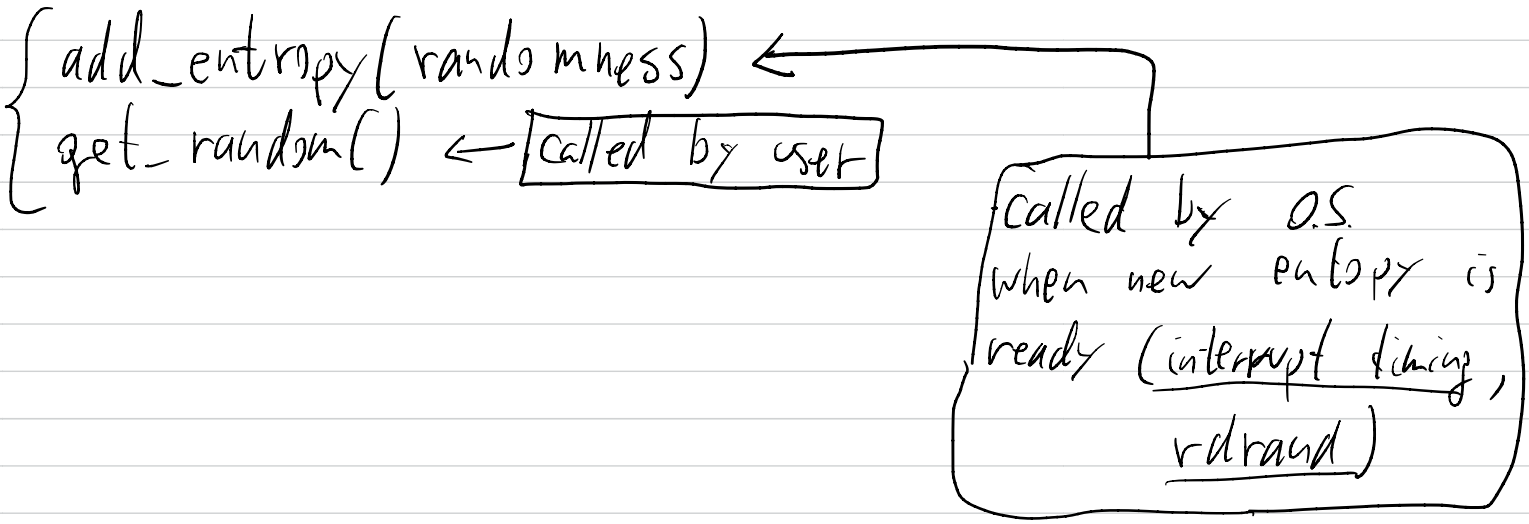
PRG: "eff" function $G: \underbrace{\{0,1\}^s}_{\text{key}} \to \underbrace{\{0,1\}^n}_{\text{output}}$, $n \gg s$

Secure PRG: every "eff" alg A behaves on the distribution $\left\{ \begin{matrix} k \xleftarrow{R} \{0,1\}^s \\ r \leftarrow G(k) \end{matrix} : \text{output } r \in \{0,1\}^n \right\}$ almost the same as on $\left\{ r \xleftarrow{R} \{0,1\}^n : \text{output } r \right\}$.

A short aside: how to generate PRF key?

how to generate cryptographic randomness?
unpredictable to attacker

random number generator (RNG) library:



Never call get_random before enough entropy added!

stream cipher: $\mathcal{M} = \mathcal{C} = \{0,1\}^{\leq n}$, $\mathcal{K} = \{0,1\}^s$

$E(k,m) = m \oplus [G(k)[0,\ldots,|m|-1]]$; $D(k,c) = c \oplus [G(k)[0,\ldots,|c|-1]]$

security: PRG secure $\Rightarrow$ stream cipher "secure"
(will define next lecture)

example PRG:

chacha20: key $\in \{0,1\}^{256}$, output: 64-byte blocks

Attacks on OTP & stream ciphers

(1) Two time pad attack:

never use a stream cipher key more than once.

example mistake: MS-PPTP

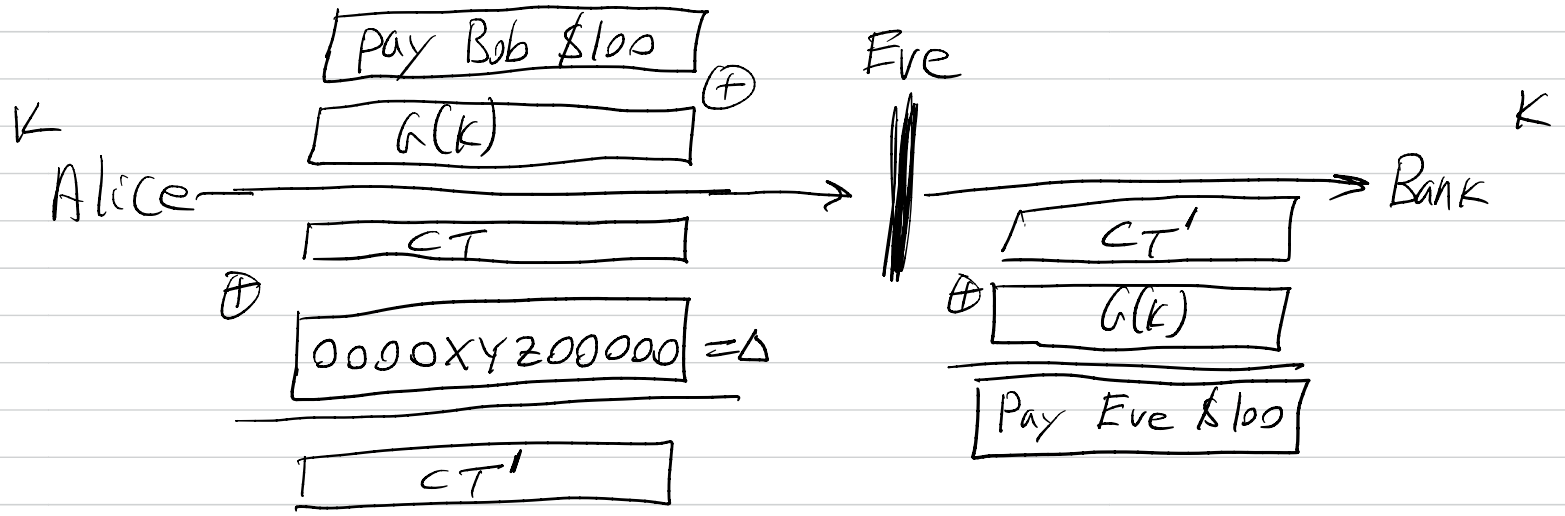
$[m_1, m_2, \ldots] \oplus G(k)$ $\qquad$ $[s_1, s_2] \oplus G(k)$

(also in 802.11b WEP)

---

(2) No integrity:

$$\underbrace{[(m \oplus G(k)) \oplus \Delta]}_{CT} \underbrace{\oplus G(k)}_{\text{decrypt}} = m \oplus \Delta$$



Bob: 42 6F 62 $\qquad$ Eve: 45 76 65 $\qquad$ XYZ = Bob ⊕ Eve = 07 19 07

---

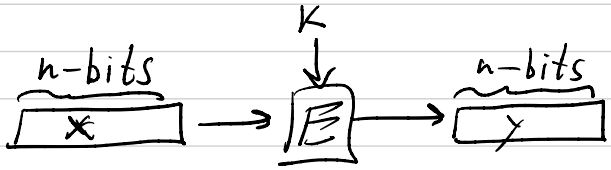
what if we want to use one key for many msgs/files?

$\Rightarrow$ block ciphers.

Block ciphers: The workhorse of cryptography

Def: A **block cipher** defined over $(\mathcal{K}, X)$ is a pair of "eff." algs. (E, D) s.t.

$E: \mathcal{K} \times X \to X$, $D: \mathcal{K} \times X \to X$ and $\forall k \in \mathcal{K}, x \in X: D(k, E(k,x)) = x$.

In a picture: 

, same for D.

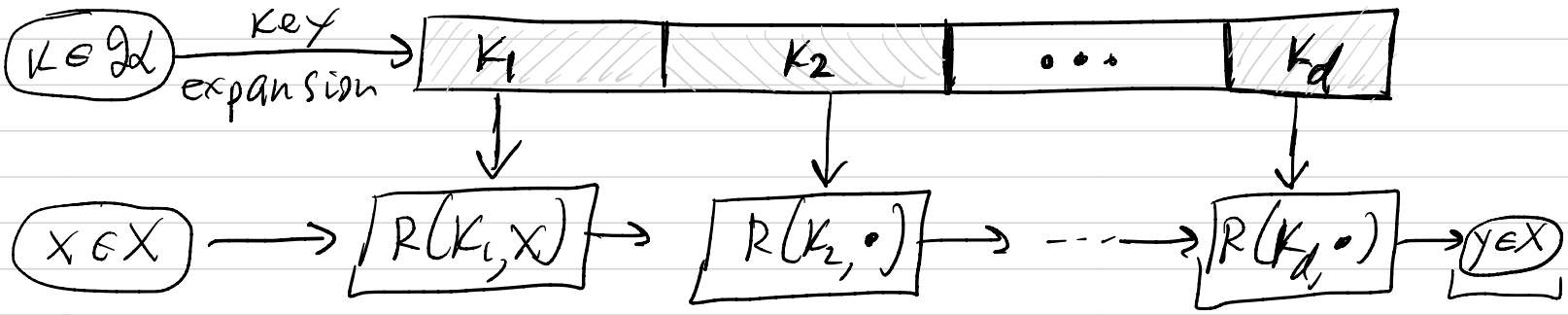
examples: (1) AES: $n = 128$, key-len = 128, 192, 256 bits

(2) 3DES: $n = 64$, key-len = 168 bits (old)

Secure block cipher: defined next lecture

intuition: for $k \xleftarrow{R} \mathcal{K}$ the function $E(x) := E(k, x)$ "looks like" a **random** one-to-one function $X \to X$

Block ciphers are built by iteration:



$R(k, x)$: round function. d = # rounds

AES128: 10 rounds,
AES192: 12 rounds,
AES256: 14 rounds

3DES: 48 rounds (slow)

(ex: iterate $x^2 + k_1 x + k_2$)

Advanced enc. std (AES - 2000)

The AES Pledge:

> I promise that once I see how simple AES really is, I will not implement it in production code even though it will be really fun. This agreement will remain in effect until I learn all about side-channel attacks and countermeasures to the point where I lose all interest in implementing AES myself.

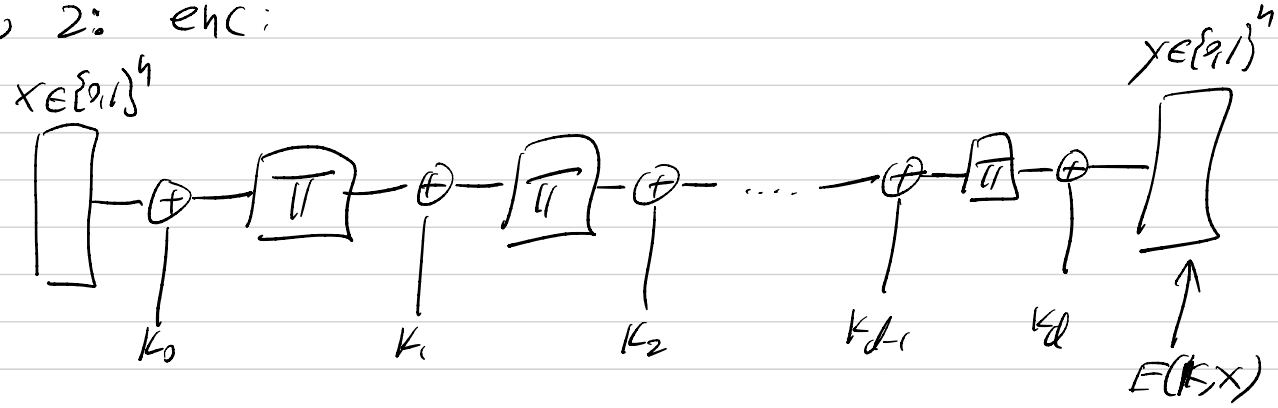
Abstract AES: Iterated Even-Mansour (IEM) cipher

IEM: $\pi : \{0,1\}^n \to \{0,1\}^n$ Fixed invertible functions ($n=128$)

cipher: alg. $E_{IEM}(K, x)$ $\quad x \in \{0,1\}^n$

step 1: key schedule $\quad K \to K_0, K_1, \ldots, K_d \in \{0,1\}^n$

step 2: enc:



alg $D_{IEM}(K, y)$: apply π^{-1} & keys in reverse order

Thm: if π is a random one-to-one function and $K_0, \ldots, K_d \xleftarrow{R} \{0,1\}^n$ then (E_{IEM}, D_{IEM}) is a secure block cipher provided n is sufficiently large.

AES:

- $n = 128$
- 128-bit key $\rightarrow$ $d = 10$
- 192 — " — $\rightarrow$ $d = 12$
- 256 — " — $\rightarrow$ $d = 14$

minor detail:
π in last round is slightly different than previous rounds.
why? makes dec. circuit same as enc. circuit

why AES256?

why not do 2AES128 with two 128-bit keys??

inefficient & insecure! HW #2

Description of π: will review next lecture

Performance

cipher	Block/key size	Speed (MB/sec)
ChaCha20	1/256	643
3DES	64/168	30
AES128	128/128	163
AES256	128/256	115

(w/o HW acceleration)

AES in hardware: AES-NI (Intel, AMD, ARM)

- aesenc, aesenclast: one round of AES.

  aesenc xmm1, xmm2 (output in xmm1)

  xmm1: state; xmm2: round key

- aesdec, aesdeclast
- aeskeygenassist: do AES key expansion

⇒ 14x speed-up over openssl on same HW.

⇒ constant time ⇒ resistant to timing attacks.

On Intel Skylake (old): 4 cycles for aesenc

Fully pipelined: can issue an inst. every cycle!

On Intel Icelake (2019): vectorized aesenc (AVX2)

- apply aesenc to 4 blocks in parallel.
- Fully pipelined.

⇒ encrypting a single block with AES128 (10 rounds) takes 40 cycles.

⇒ encrypting 16 blocks (in parallel) on Icelake takes 43 cycles