Lecture 4

Pseudo-random Generator (PRG)

$y = PRG(x)$

- output, keystream: $y \in \{0,1\}^*$
- input, seed: $x \in \{0,1\}^k$
  - ↳ $k \geq 112$ bits
- deterministic

Security Def'n

A PRG is secure if it is infeasible to distinguish the output y from a truly random bitstring of the same length (if you don't know x)

- ↳ Lemma: no way to recover x from some output y.
- ↳ Lemma: infeasible to predict the next bit in y given all the preceding bits (but not x)

Examples

Non-crypto: LFSRs, Mersenne Twister

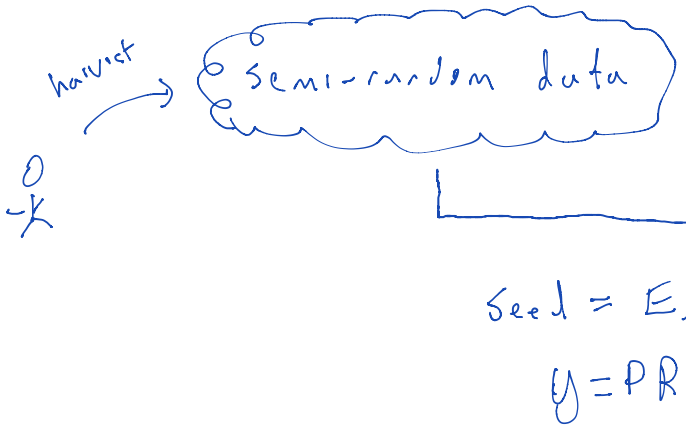
Crypto: (1) Inside a stream cipher
- ↳ A5/1, A5/2 → insecure
- ↳ RC4 → insecure
- ↳ Chacha 20 → secure

(2) Can use a block cipher in counter-mode (e.g. AES-CTR)

Use Cases

(1) Encryption → coming soon

(2) Computer Randomness



Example:

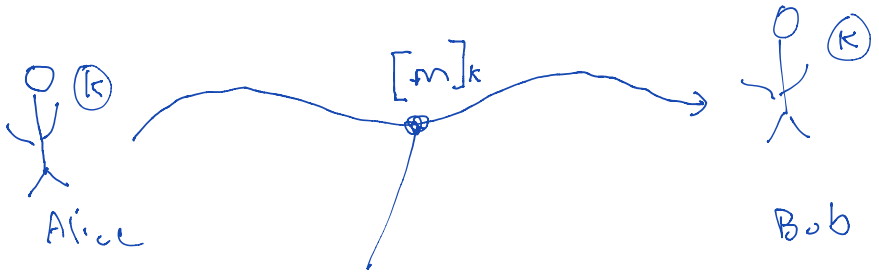
hexdump /dev/random (Unix)
- ↳ access to Ext(bitstream) but it might halt.

/dev/urandom
- ↳ output of the PRG, never halt

Review

	x ← input	y ← output.	
Hash	$\{0,1\}^*$	$\{0,1\}^d$	PR, CR
Ext	$\{0,1\}^*$	$\{0,1\}^d$	$\lvert x\rvert \geq 2d$, y is statistically random
PRG	$\{0,1\}^k$	$\{0,1\}^*$	one-way, y is computationally random

Symmetric key Encryption



↳ does not know k

↳ infeasible to learn any information about m.

Symmetric key Encryption
- → one-time pad
- → stream cipher
- → block cipher

Note: Punt on the question of how Alice and Bob end up with the same secret key k

↳ see later in the course

Functions

$c = Enc(k, m)$

- $k \in \{0,1\}^*$ — key (shared secret)
- $m \in \{0,1\}^*$ — message (secret)
- $c \in \{0,1\}^*$ — ciphertext
- $|c| = |m|$
- deterministic, open specification

$m = Dec(k, c)$

- same key as encryption → secret.

One-time Pad

* Theoretical construction (not practical)
* Perfectly secure
  - ↳ break it is not just infeasible, it is impossible.
* Drawback: key size is the same as the message.
* Algorithm: for each bit i of the message:

  Encryption: $c_i = Enc_{k_i}(m_i) = k_i \oplus m_i$

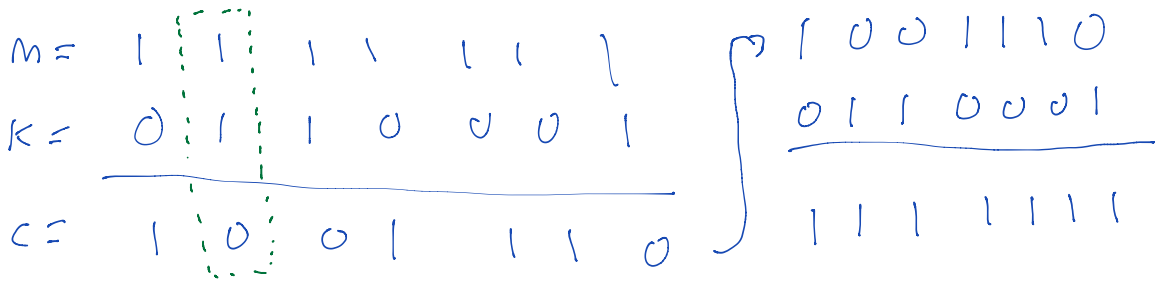
  - k_i: truly random.
  - $\oplus$: xor
  - m_i: message bit

$m_i = Dec_{k_i}(c_i) = k_i \oplus c_i$

xor

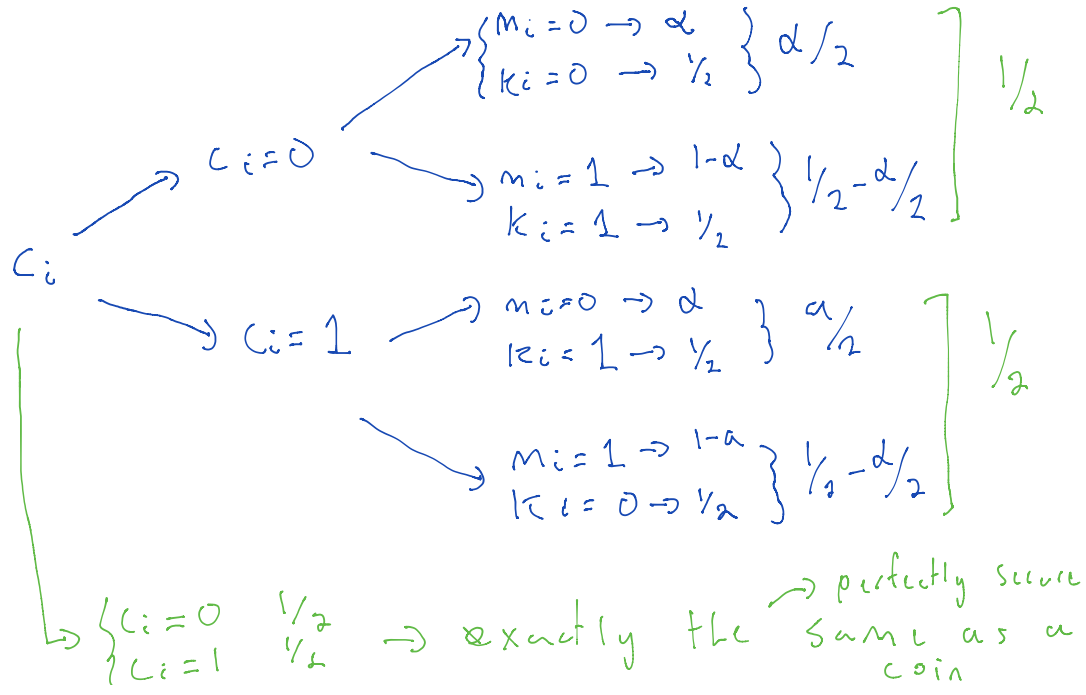
	0	1
0	0	1
1	1	0

or $a+b \bmod 2$



Security

If Eve sees c, it is impossible to learn anything about m without knowing k (except the length of m)



Drawbacks

1. k is large and both parties need it
2. k needs to be truly random
3. Cannot reuse bits of k.

$$\left.\begin{aligned} m_i \oplus k_i &= c_i \\ m_i' \oplus k_i &= c_i' \end{aligned}\right\} c_i \oplus c_i' = m_i \oplus m_i' \oplus k_i \oplus k_i = m_i \oplus m_i'$$

* don't learn m_i or m_i' but learn if they are the same or different

4. Malleability.
   - ↳ Assume Eve sees c but not m or k
   - ↳ if Eve flips c_i then when Bob decrypts, m_i will be flipped

5. Leaks $|m_i|$

Stream Cipher

Solve ① and ②

③ → won't solve, be careful

④ → will solve eventually with MACs

⑤ → still an issue today!

Reconsider OTP: $C = M \oplus K$

↳ bitwise xor

Stream Cipher:

$C = Enc(m,k) = m \oplus PRG(k)$

$|m|$, $|m|$, $\{0,1\}^{|m|}$

≥ 112-bit seed

$m = Dec(c,k) = c \oplus PRG(k)$

↳ bitwise xor

Security:

$C = Enc(m,k) = m \oplus PRG(k)$

↳ indistinguishable from random → indist. from a OTP

* If you know m, you can compute $PRG(k)$ from c however

a) This tells you nothing about k

b) This tells you nothing about the next bits from $PRG(k)$

Chacha 20

* Dan Bernstein
* Used by Google (chrome ⟷ google services)
* Not NIST
* See Description (not tested)
  - ↳ do the same basic tasks lots of times
  - ↳ simple, logic-based → fast
  - ↳ confusion → add nonlinearity that is hard to reverse
  - ↳ diffusion → spread the confusion

Block Cipher

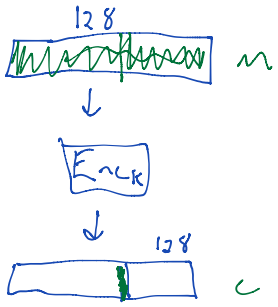
$\{0,1\}^k$ $k \geq 112$

$\{0,1\}^\ell$

$$c = Enc(k, m)$$

$\{0,1\}^\ell$

$$m = Dec_k(k, c)$$



Enc works on a fixed length message of ℓ-bits (generally 128 or 256 bits) called a block

We want to encrypt any size data so we need to combine encryption functions in some structure → mode of operation.