CSCI 1510

- Merkle-Damgård Transform
- Hash-and-MAC
- Applications of Hash Functions
- Constructions of Block Cipher

Collision-Resistant Hash Function (CRHF)

- **Syntax:**

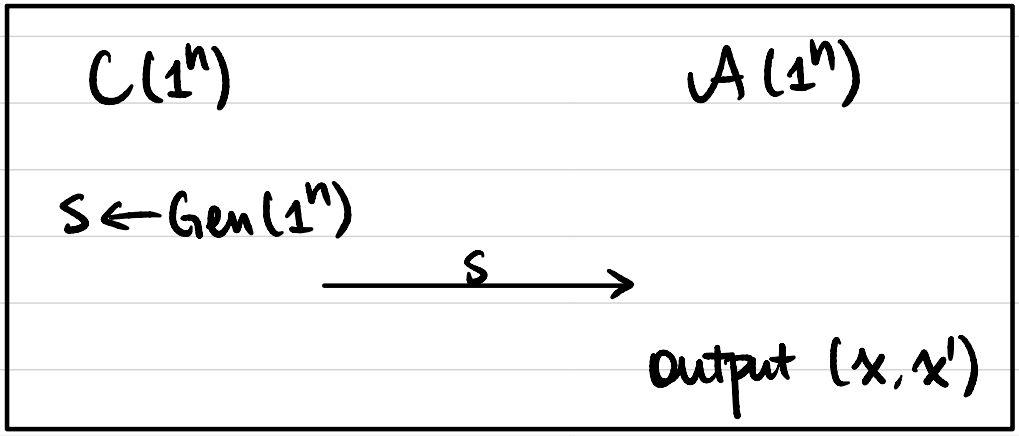
  A hash function is defined by a pair of PPT algorithms (Gen, H):

  - $Gen(1^n)$: output s
  - $H^s(x)$: $x \in \{0,1\}^*$, output $h \in \{0,1\}^{\ell(n)}$

- **Security**

  A hash function (Gen, H) is **collision-resistant** if

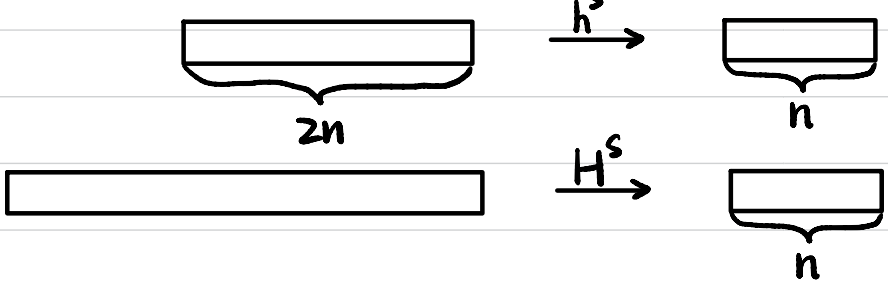
  $\forall$ PPT $\mathcal{A}$, $\exists$ negligible function $\varepsilon(\cdot)$ s.t. $\Pr[x \neq x' \wedge H^s(x) = H^s(x')] \leq \varepsilon(n)$.



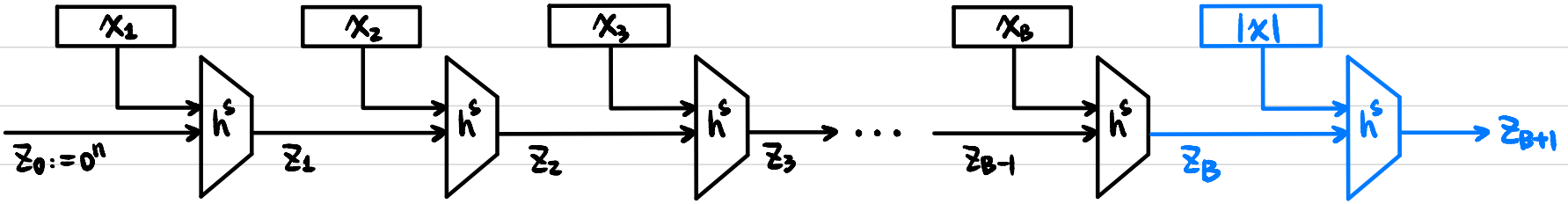
Domain Extension: Merkle-Damgård Transform

Given a CRHF (Gen, h) from $\{0,1\}^{2n}$ to $\{0,1\}^n$.

Construct a CRHF (Gen, H) from $\{0,1\}^*$ to $\{0,1\}^n$.



- $Gen(1^n)$: remains unchanged.
- $H^s(x)$: $x \in \{0,1\}^*$
  - ① Pad x with $100\cdots0$ to a multiple of n $\rightarrow \tilde{x}$
  - ② Parse $\tilde{x} = x_1 \| x_2 \| \cdots \| x_B$, $x_i \in \{0,1\}^n$ $\forall i \in [B]$



bit representation of $|x|$

$z_0 := 0^n$ $\quad z_i := h^s(z_{i-1} \| x_i)$ $\forall i \in [B]$ $\quad z_{B+1} := h^s(z_B \| \langle |x| \rangle)$ $\quad H^s(x) := z_{B+1}$

Thm If (Gen, h) is CRHF, then so is (Gen, H).

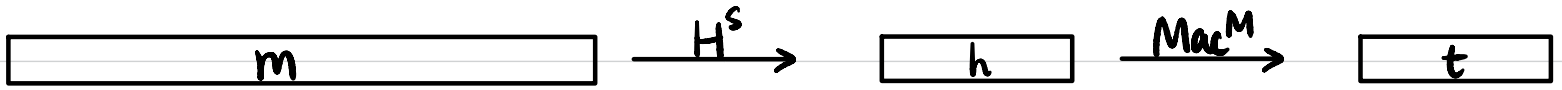
Hash-and-MAC

Secure MAC for fixed-length messages
+
CRHF for arbitrary-length inputs
$\Rightarrow$ Secure MAC for arbitrary-length messages

Let $\Pi^M = (Gen^M, Mac^M, Vrfy^M)$ be a secure MAC for messages of length n.
Let $\Pi^H = (Gen^H, H)$ be a CRHF for arbitrary-length inputs with output length n.

Construct $\Pi = (Gen, Mac, Vrfy)$:

- $Gen(1^n)$: $k^M \leftarrow Gen^M(1^n)$. $s \leftarrow Gen^H(1^n)$. Output $k = (k^M, s)$
- $Mac(k, m)$: $m \in \{0,1\}^*$. Parse $k = (k^M, s)$
  $h := H^s(m)$, $t \leftarrow Mac^M(k^M, h)$. Output t.
- $Vrfy(k, (m, t))$: Parse $k = (k^M, s)$
  $h := H^s(m)$. $b := Vrfy^M(k^M, (h, t))$. Output b.



Thm If Π^M is a secure MAC and Π^H is CRHF, then Π is a secure MAC.

Applications of Hash Functions

- **Deduplication**

$H(D_1) \rightarrow h_1$

$H(D_2) \rightarrow h_2$

unique identifier

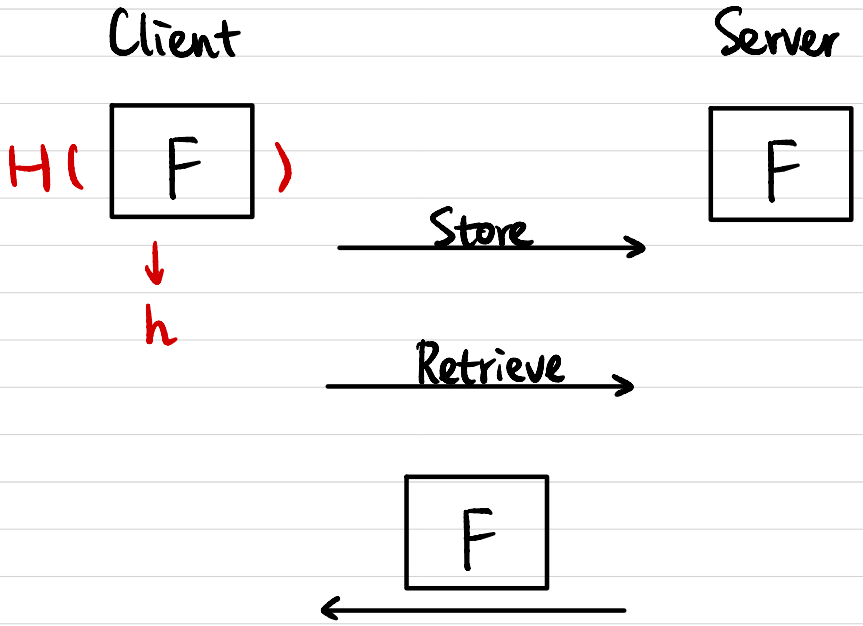
If $h_1 \neq h_2 \Rightarrow D_1 \neq D_2$

If $h_1 = h_2 \Rightarrow D_1 = D_2$ Why?

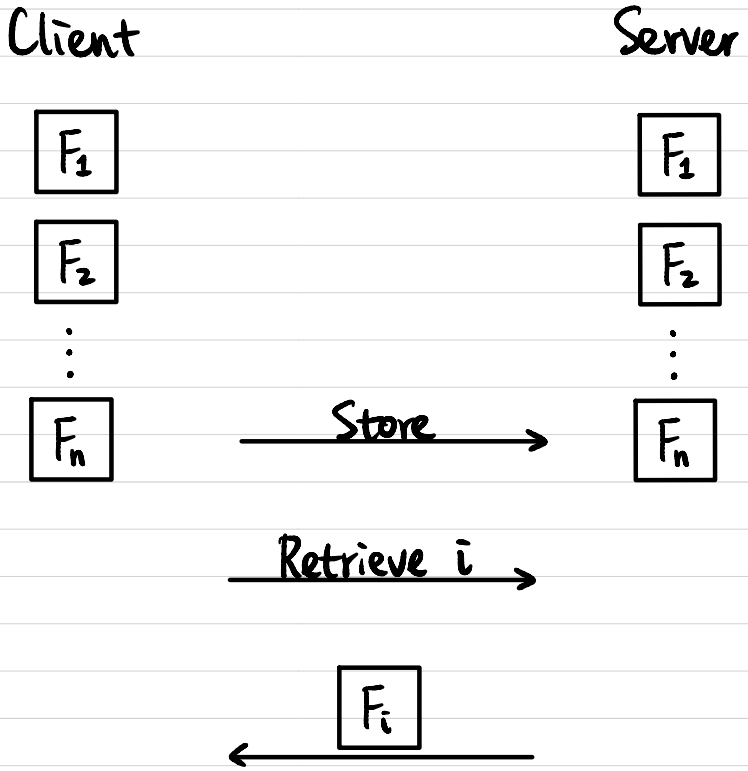
Virus Scan $H(F) \stackrel{?}{=} H(F^*)$

Video Deduplication $H(V_1) \stackrel{?}{=} H(V_2)$

Applications of Hash Functions



Is the file changed?



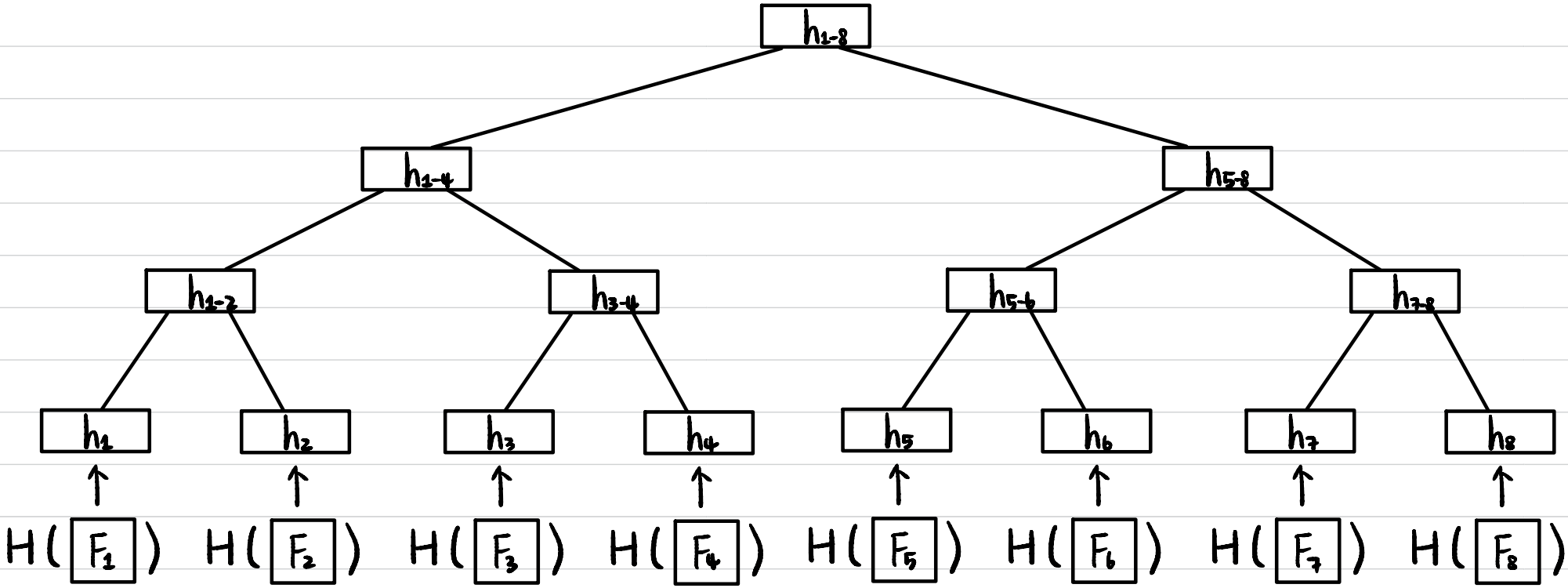
Is the file changed?

Goal:

① Client's storage doesn't grow with n.

② Verification doesn't grow with n.

Merkle Tree

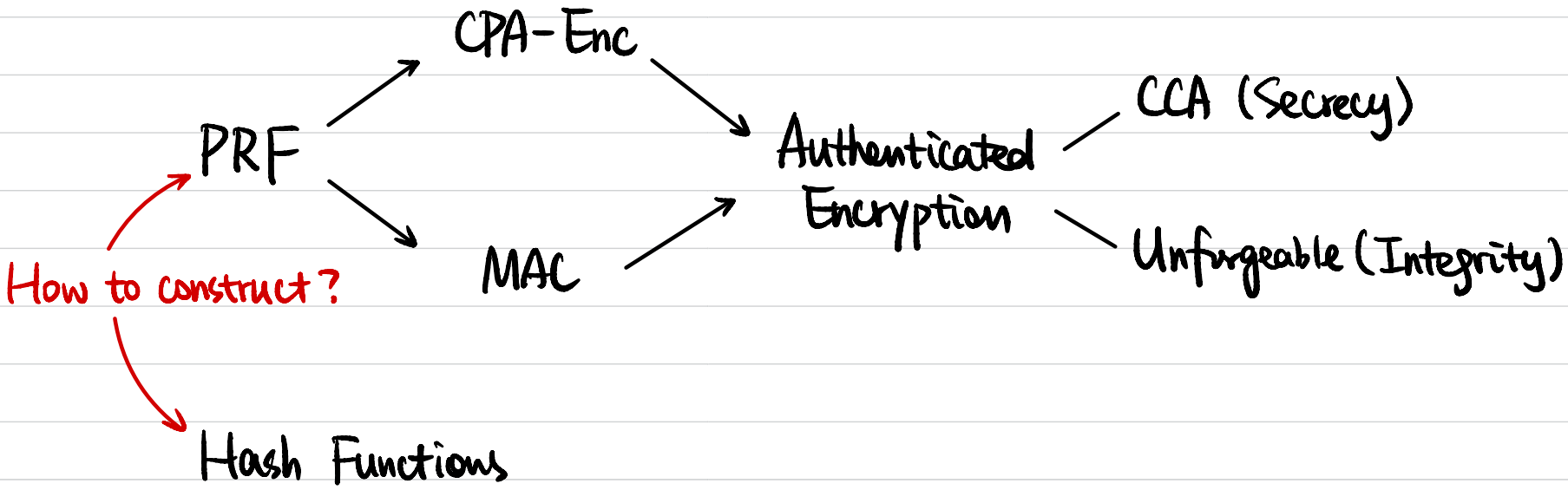


$H^s: \{0,1\}^* \to \{0,1\}^n$

$MT_t^s(F_1 \| \cdots \| F_t) \to \{0,1\}^n$

How does verification work?

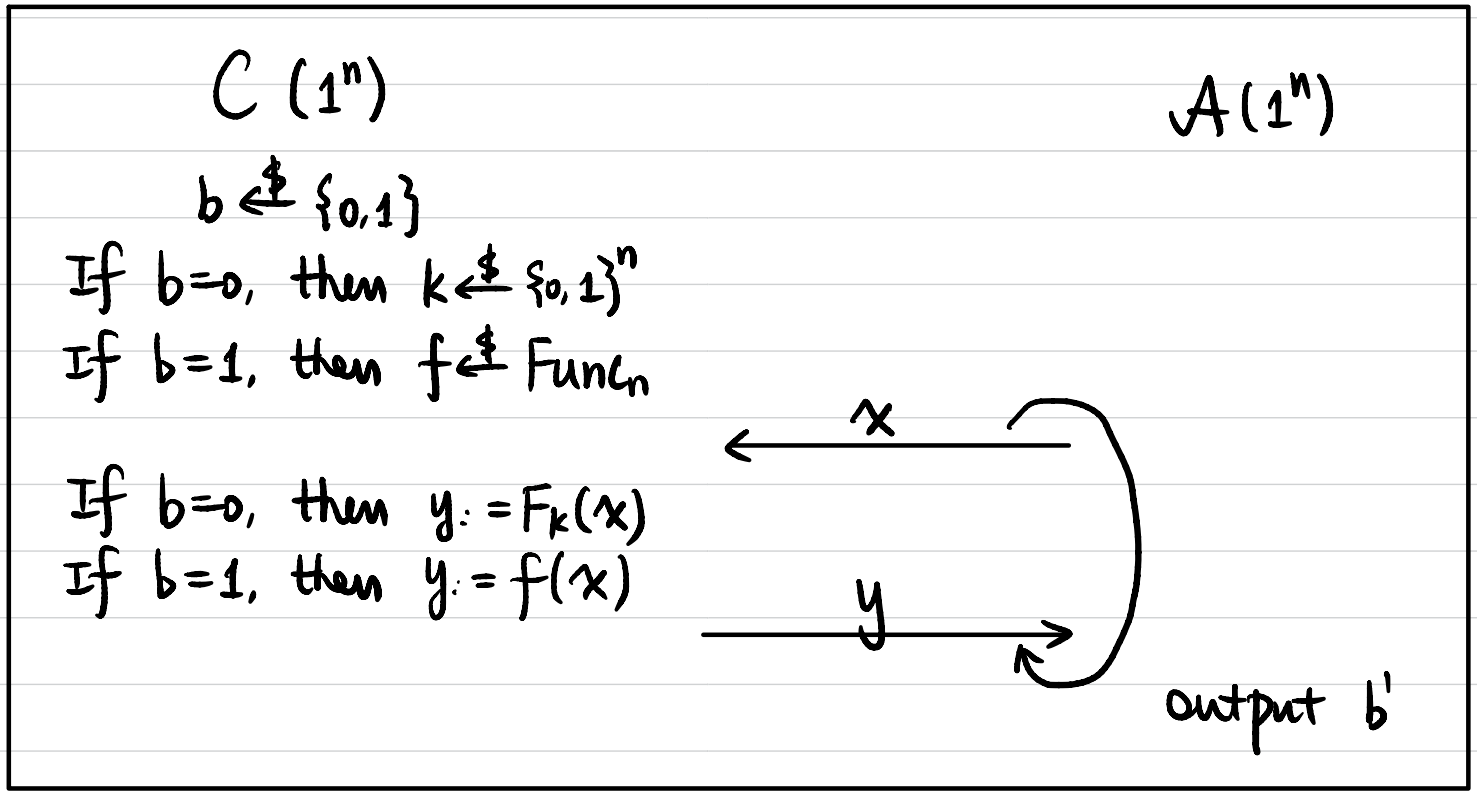
Thm If (Gen, H) is a CRHF, then (Gen, MT_t) is a CRHF for any fixed $t = 2^k$.


CPA-Enc
PRF
MAC
Authenticated Encryption
CCA (Secrecy)
Unforgeable (Integrity)
How to construct?
Hash Functions

Pseudorandom Function (PRF)

Def Let $F: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a deterministic, poly-time, keyed function. F is a pseudorandom function (PRF) if $\forall$ PPT A, $\exists$ negligible function $\varepsilon(\cdot)$ s.t.

$$\left| \Pr_{k \leftarrow U_n}[A^{F_k(\cdot)}(1^n) = 1] - \Pr_{f \xleftarrow{\$} \text{Func}_n}[A^{f(\cdot)}(1^n) = 1] \right| \leq \varepsilon(n)$$

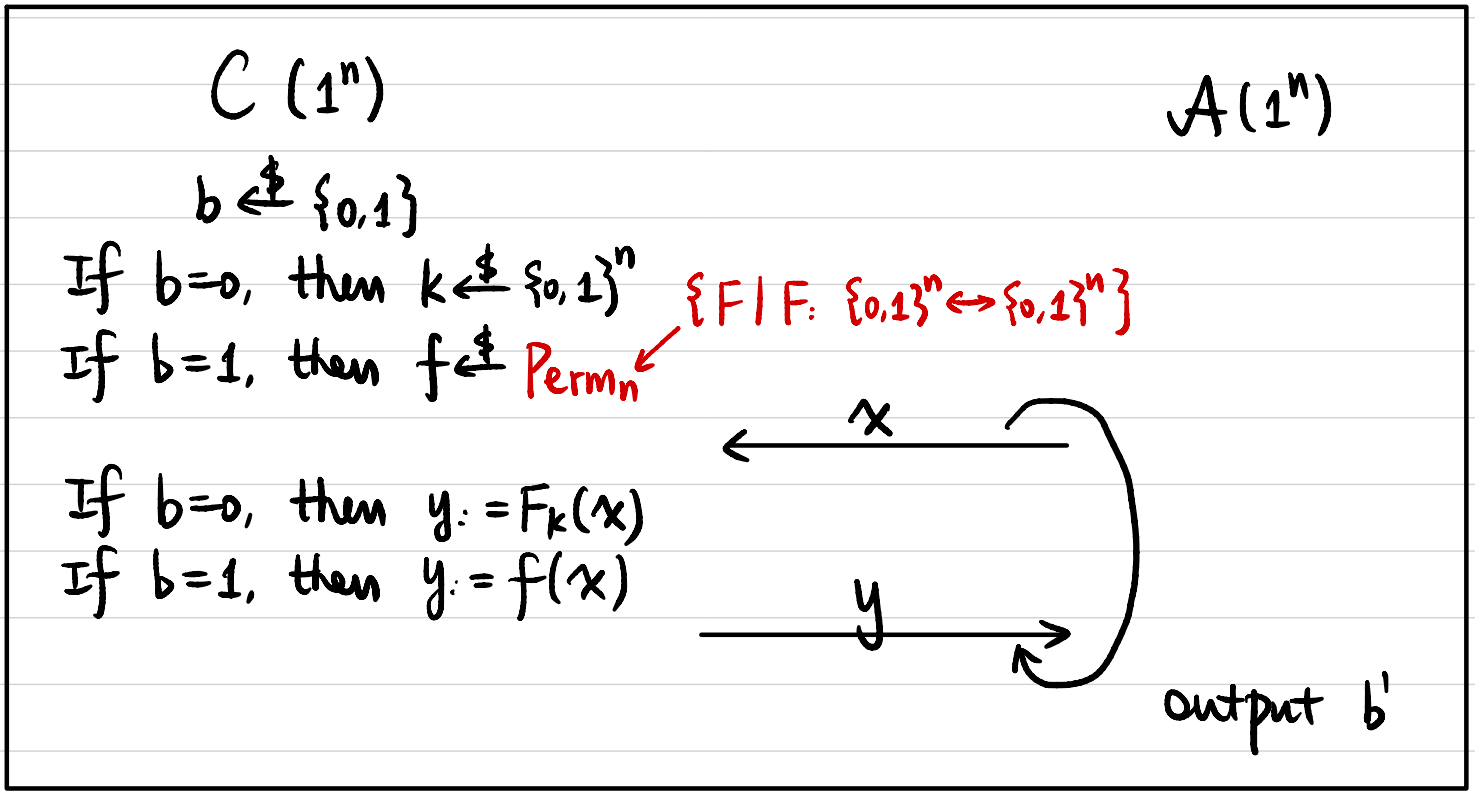


$$\Pr[b = b'] \leq \frac{1}{2} + \varepsilon(n).$$

Pseudorandom Permutation (PRP)

Def Let $F: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a deterministic, poly-time, keyed function. F is a **pseudorandom permutation (PRP)** if $F_k(\cdot)$ is bijective for all k, $\forall$ PPT A, $\exists$ negligible function $\varepsilon(\cdot)$ s.t.

$$\left| \Pr_{k \leftarrow U_n}[A^{F_k(\cdot)}(1^n) = 1] - \Pr_{f \xleftarrow{\$} Perm_n}[A^{f(\cdot)}(1^n) = 1] \right| \leq \varepsilon(n)$$



$$\Pr[b = b'] \leq \frac{1}{2} + \varepsilon(n).$$

Block Cipher

$F: \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$

n: key length

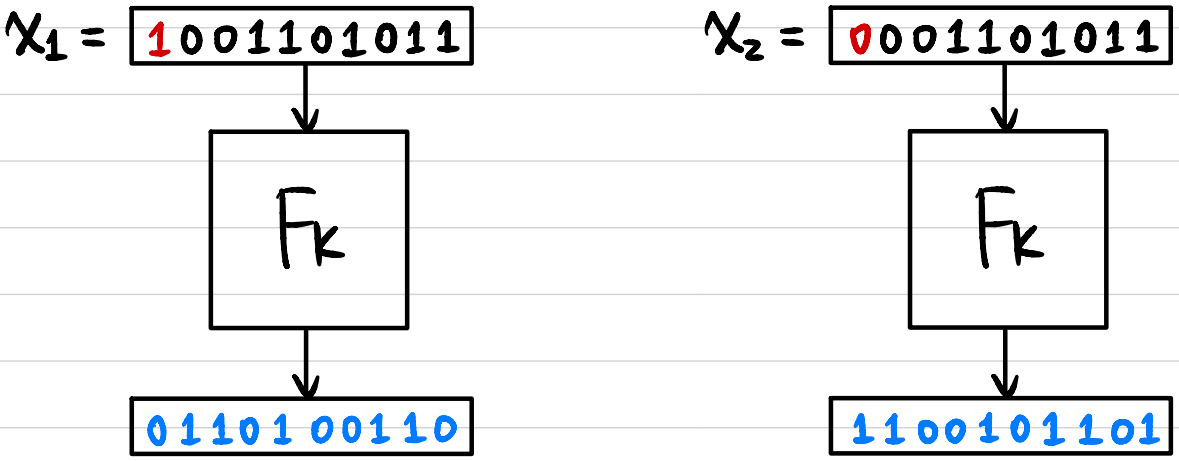
l: block length

$F_k(\cdot)$: permutation / bijective $\{0,1\}^l \to \{0,1\}^l$

$F_k^{-1}(\cdot)$: efficiently computable given k.

Assumed to be a pseudorandom permutation (PRP).

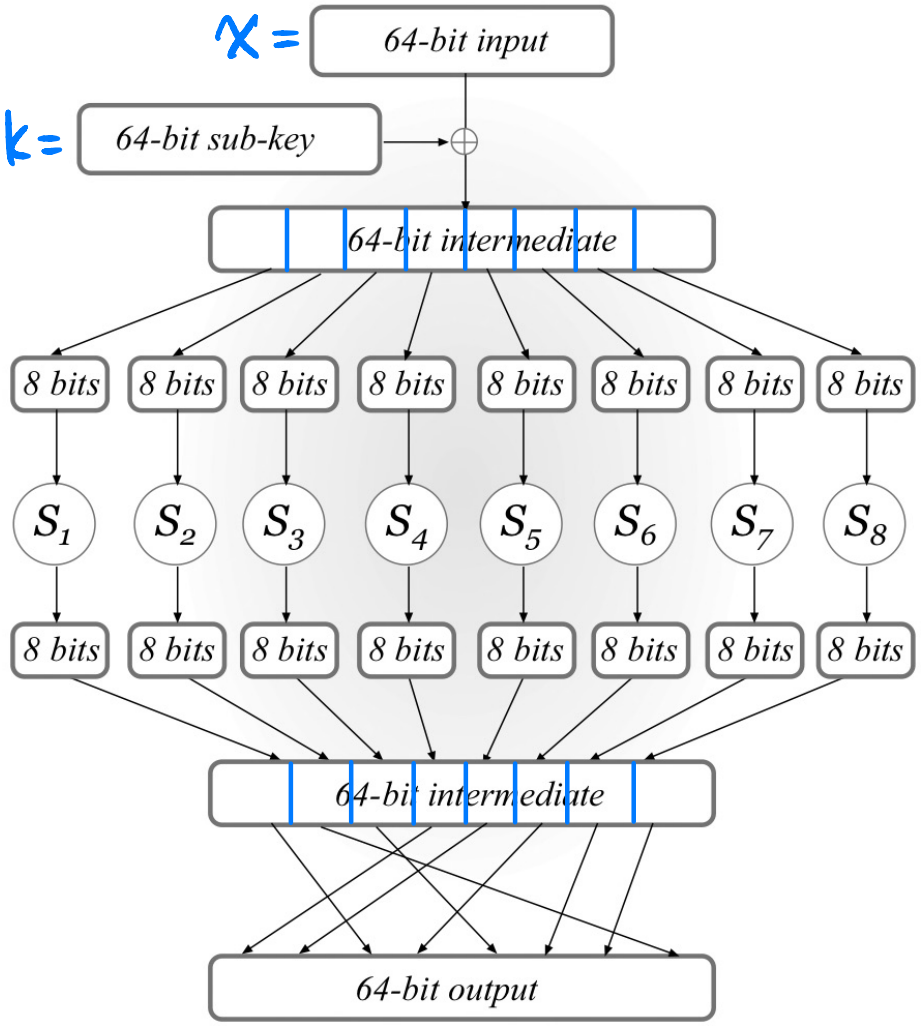
Substitution-Permutation Network (SPN)



Design Principle: "Avalanche Effect"

A one-bit change in the input should "affect" every bit of the output.

Substitution-Permutation Network (SPN)



A single round of SPN

"Confusion-Diffusion Paradigm"

Step 1: Key Mixing

$X := x \oplus k$

Step 2: Substitution (Confusion Step)

$S_i : \{0,1\}^8 \to \{0,1\}^8$ (S-box)

Public permutation / one-to-one map

1-bit change of input
→ at least 2-bit change of output

Step 3: Permutation (Diffusion Step)

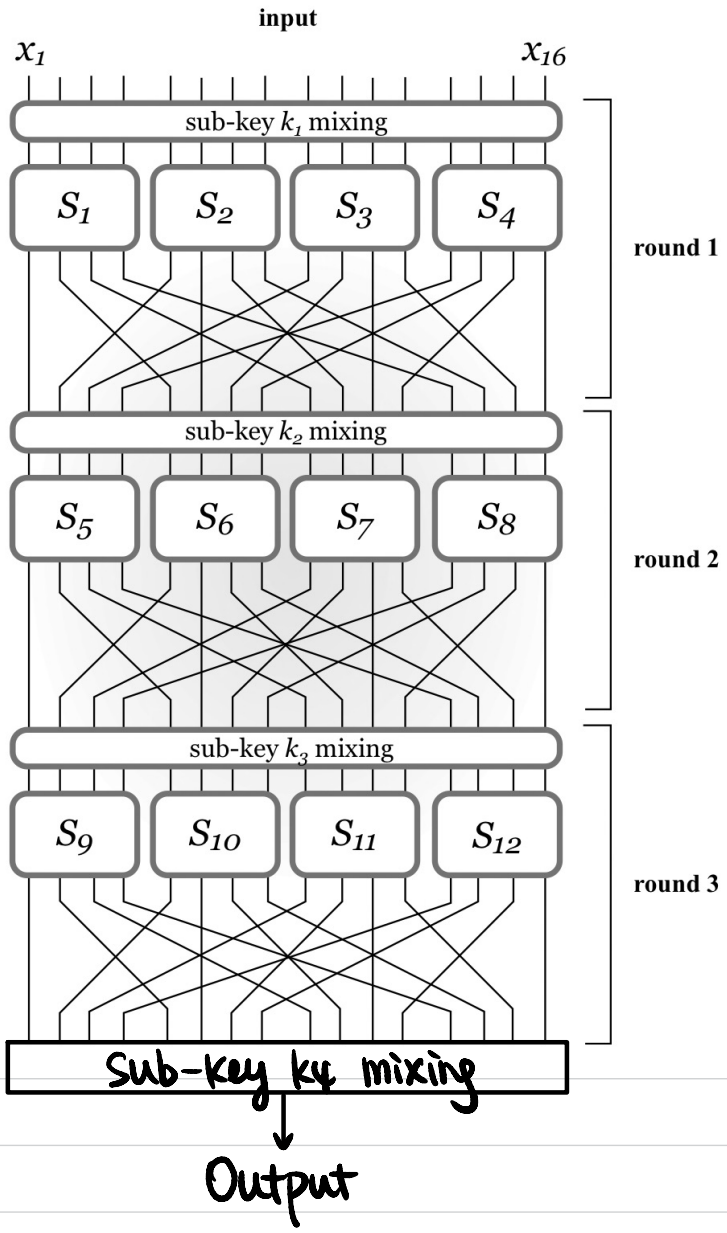
$P : [64] \to [64]$

Public mixing permutation

↓

affect input to multiple S-boxes next round

Substitution-Permutation Network (SPN)



3-round SPN:

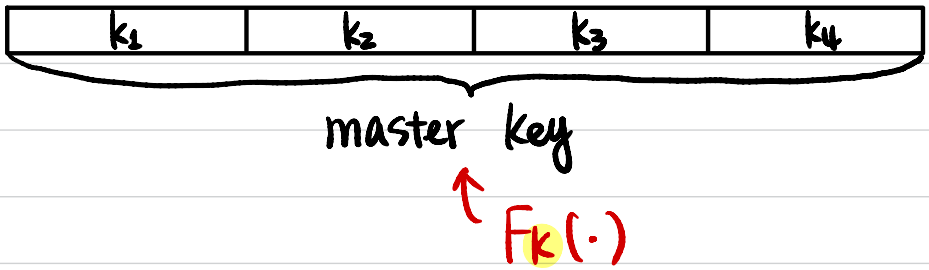
3-round
- key mixing
- substitution
- permutation

1 final-round key mixing

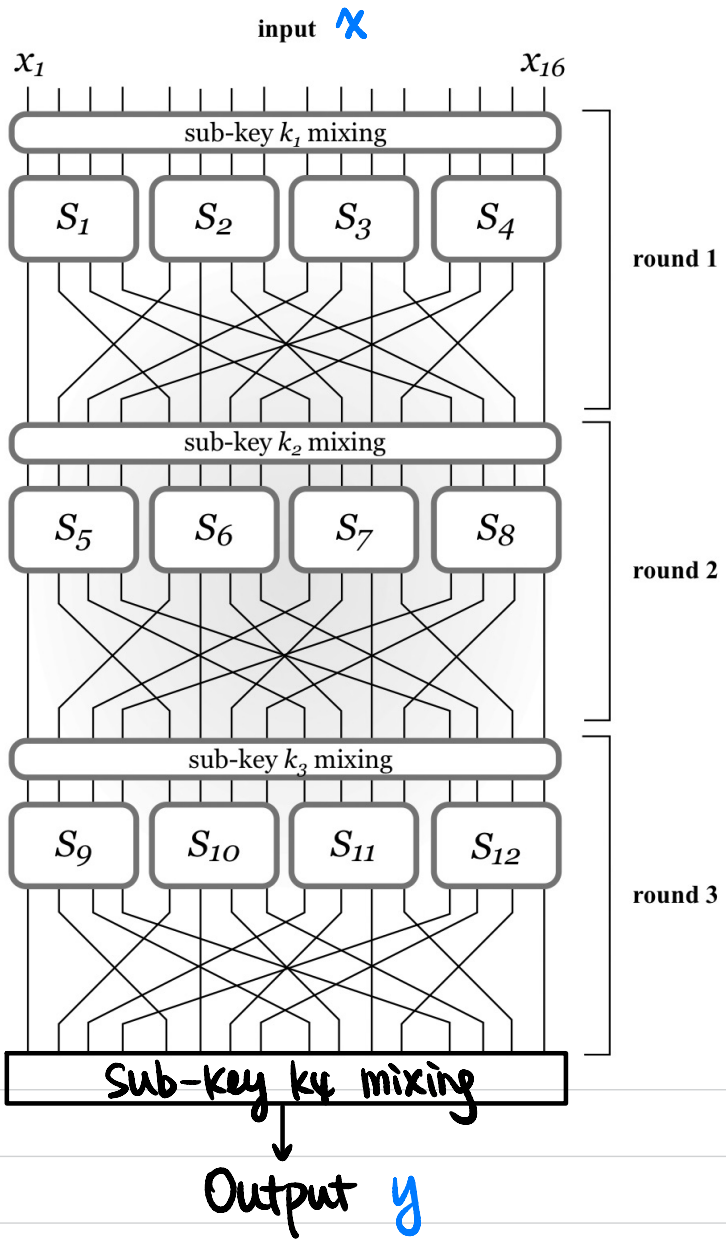
Key Schedule:

How we derive sub-keys from master key.

Example:



Substitution-Permutation Network (SPN)



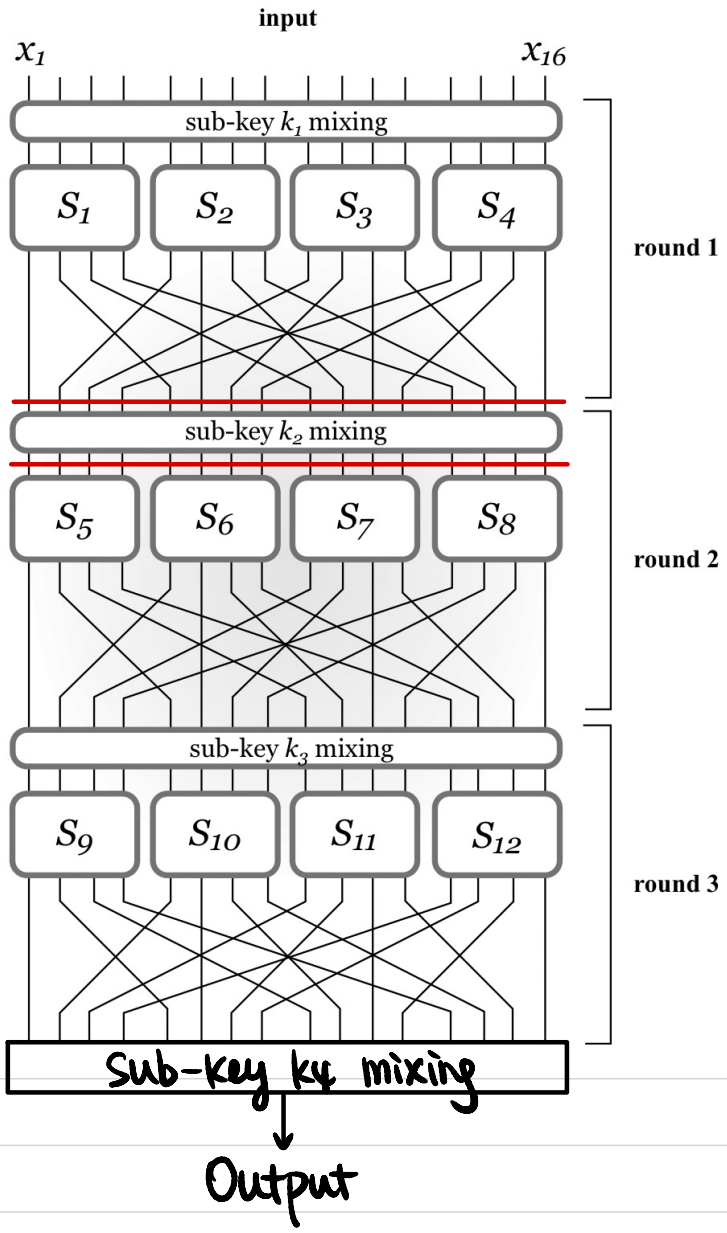
An SPN is invertible given the master key.

⇓

permutation

How to compute $F_K^{-1}(y)$?

Attacks on Reduced-Round SPN



1-round SPN without final key mixing?

1-round SPN with final key mixing?

Why do we need a final key mixing step?

Can we do r-round key mixing, then r-round substitution, then r-round permutation?