

CSCI 1515 Applied Cryptography

This Lecture:

- Practical Constructions of Block Cipher (Continued)
- Secure Hardware: Intel SGX, HSM
- Blockchain

Block Cipher

$$F: \{0, 1\}^\lambda \times \{0, 1\}^n \rightarrow \{0, 1\}^n$$

λ : key length

n : block length

It is assumed to be a pseudorandom permutation (PRP).

Construction: Advanced Encryption Standard (AES)

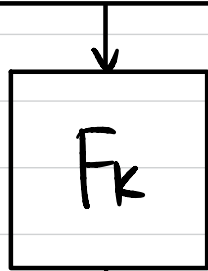
- $\lambda = 128/192/256$, $n = 128$
- Standardized by NIST in 2001
- Competition 1997-2000

Before AES: Data Encryption Standard (DES)

- $\lambda = 56$, $n = 64$

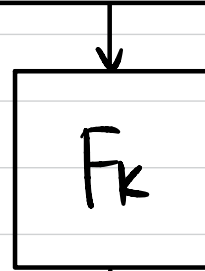
Substitution-Permutation Network (SPN)

$X_1 =$ 1001101011



0110100110

$X_2 =$ 0001101011

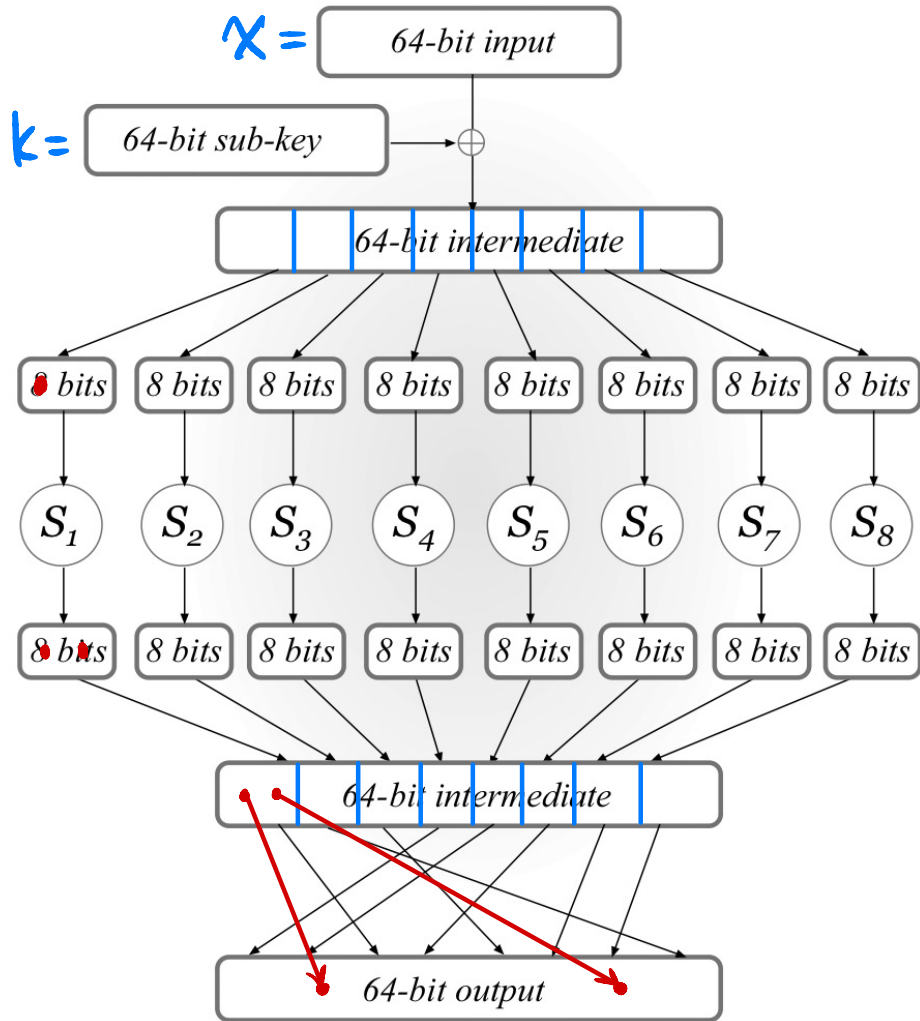


1100101101

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)

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

Public permutation/one-to-one map

1-bit change of input

→ at least 2-bit change of output

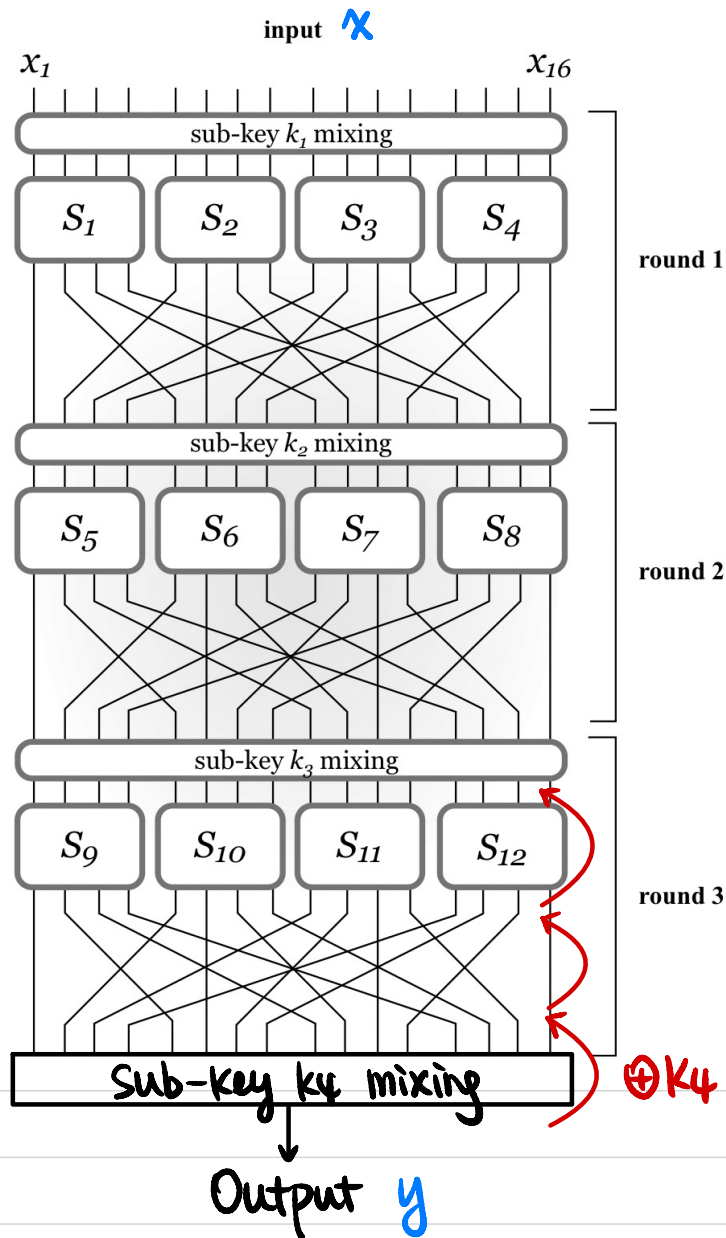
Step 3: Permutation (Diffusion Step)

$$P: [64] \rightarrow [64]$$

Public mixing permutation

↓
affect input to multiple S-boxes next round

Substitution-Permutation Network (SPN)



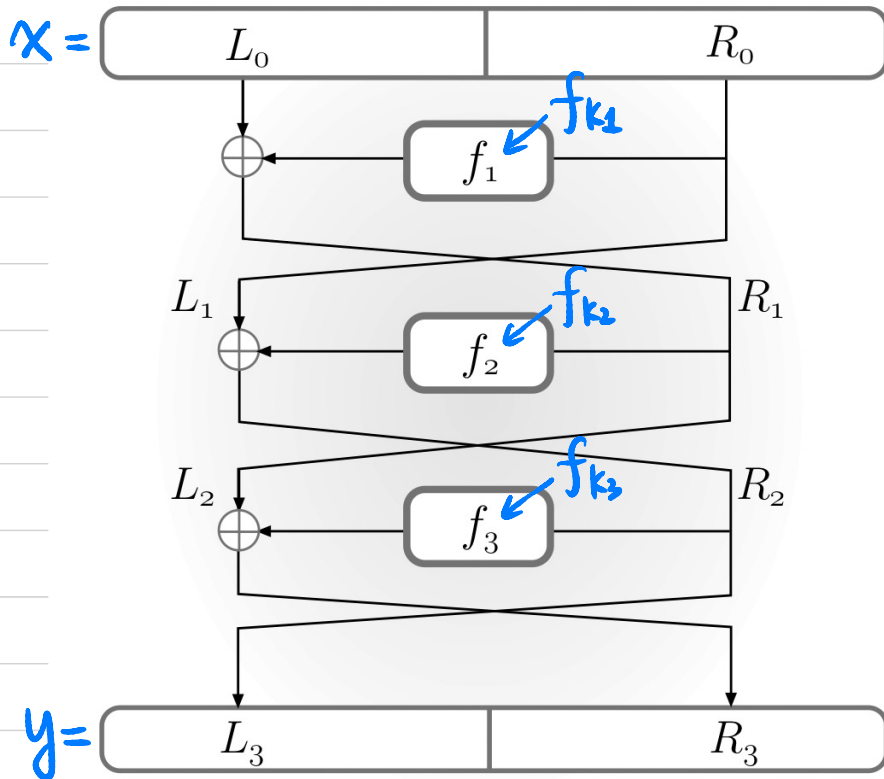
An SPN is invertible given the master key.

↓
permutation

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

Why do we need a final key mixing step?

Feistel Network



3-round Feistel Network

$f_{k_i}: \{0,1\}^{n/2} \rightarrow \{0,1\}^{n/2}$
↑
round function

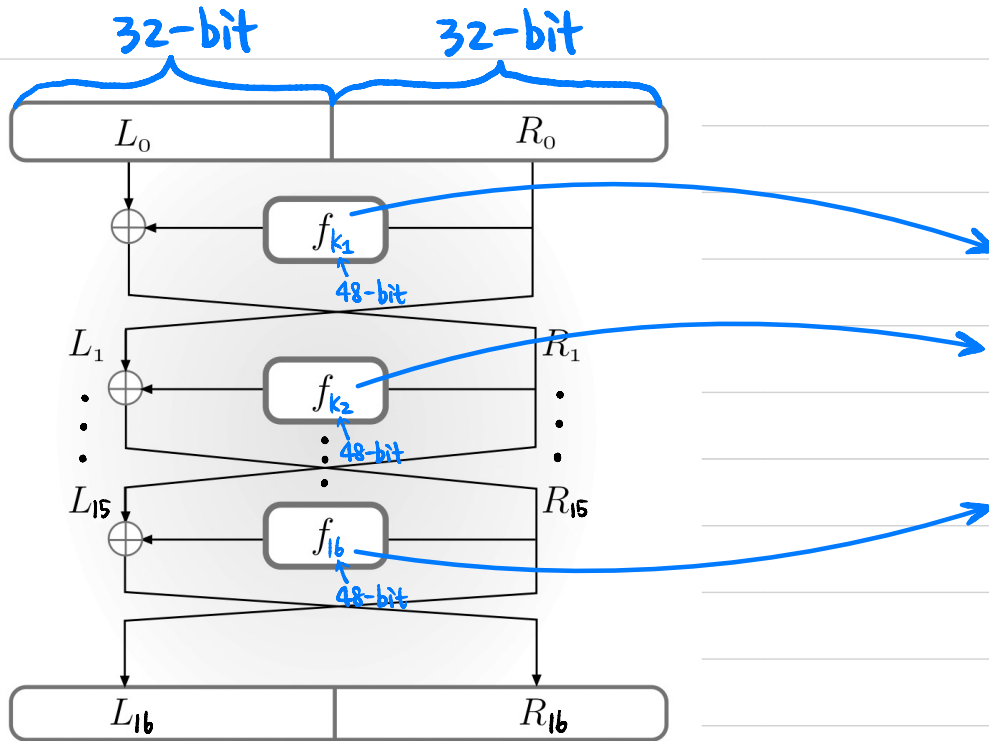
How to compute $F_k^{-1}(y)$?

Attacks on reduced-round Feistel Network

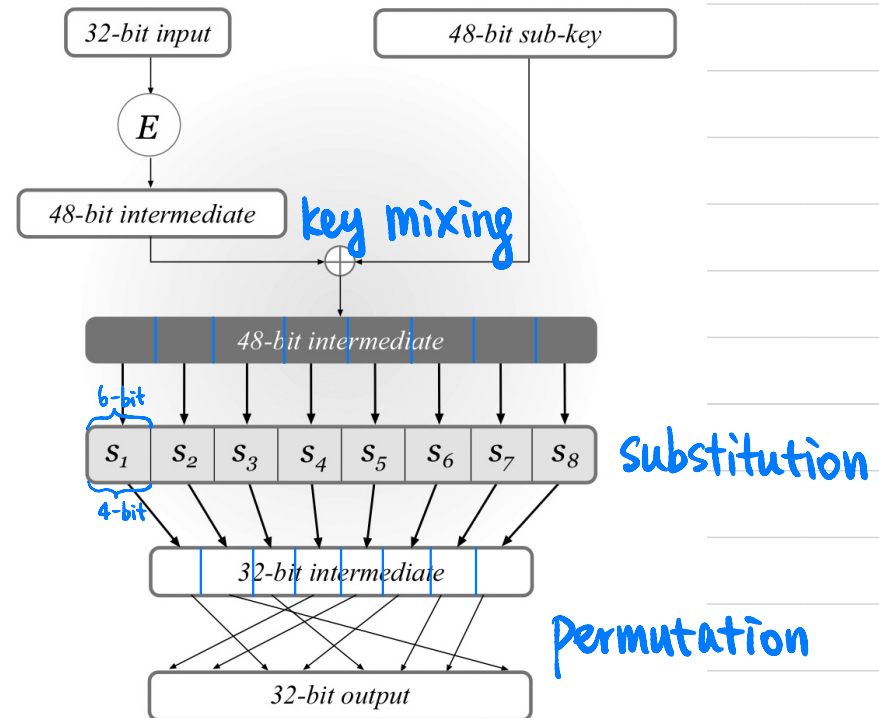
Data Encryption Standard (DES)

$F: \{0, 1\}^{\lambda} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$
block length $n=64$
master key length $\lambda=56$

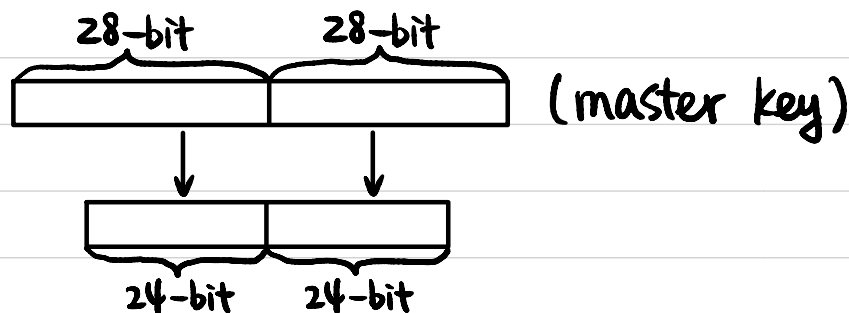
16-round Feistel Network



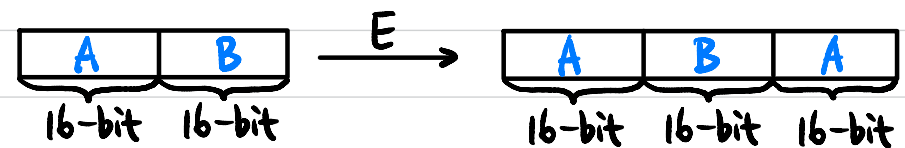
DES mangler function



Key Schedule:

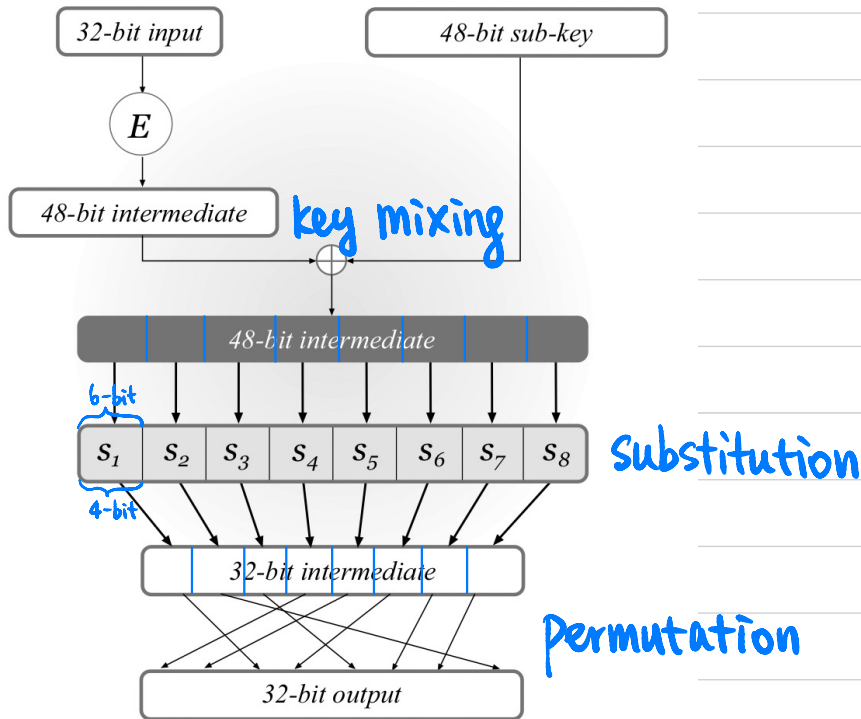


E: expansion function



Data Encryption Standard (DES)

DES mangler function



S-box: $\{0,1\}^6 \rightarrow \{0,1\}^4$

① "4-to-1":

Exactly 4 inputs map to same output

② 1-bit change of input

→ at least 2-bit change of output

Mixing Permutation: $[32] \rightarrow [32]$

4 bits from each S-box will affect the input to 6 S-boxes in the next round

Outsourcing Computation by FHE

Server



ct

f



Memory



cpu



Public
Eval

Client



Data x

Key sk

$ct \leftarrow \text{Enc}(x)$

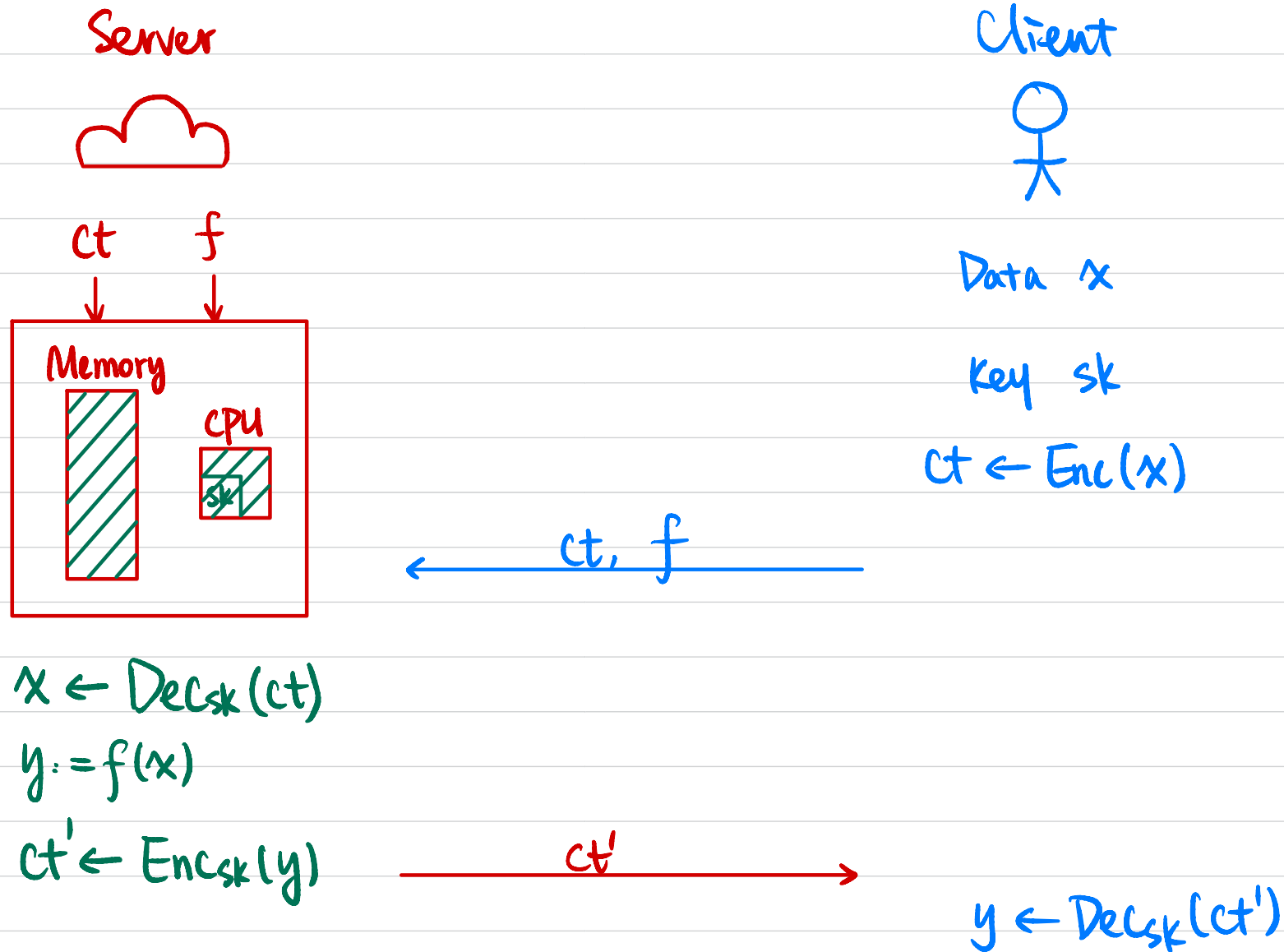
$\leftarrow ct, f$

$ct' \leftarrow \text{Eval}(f, ct)$

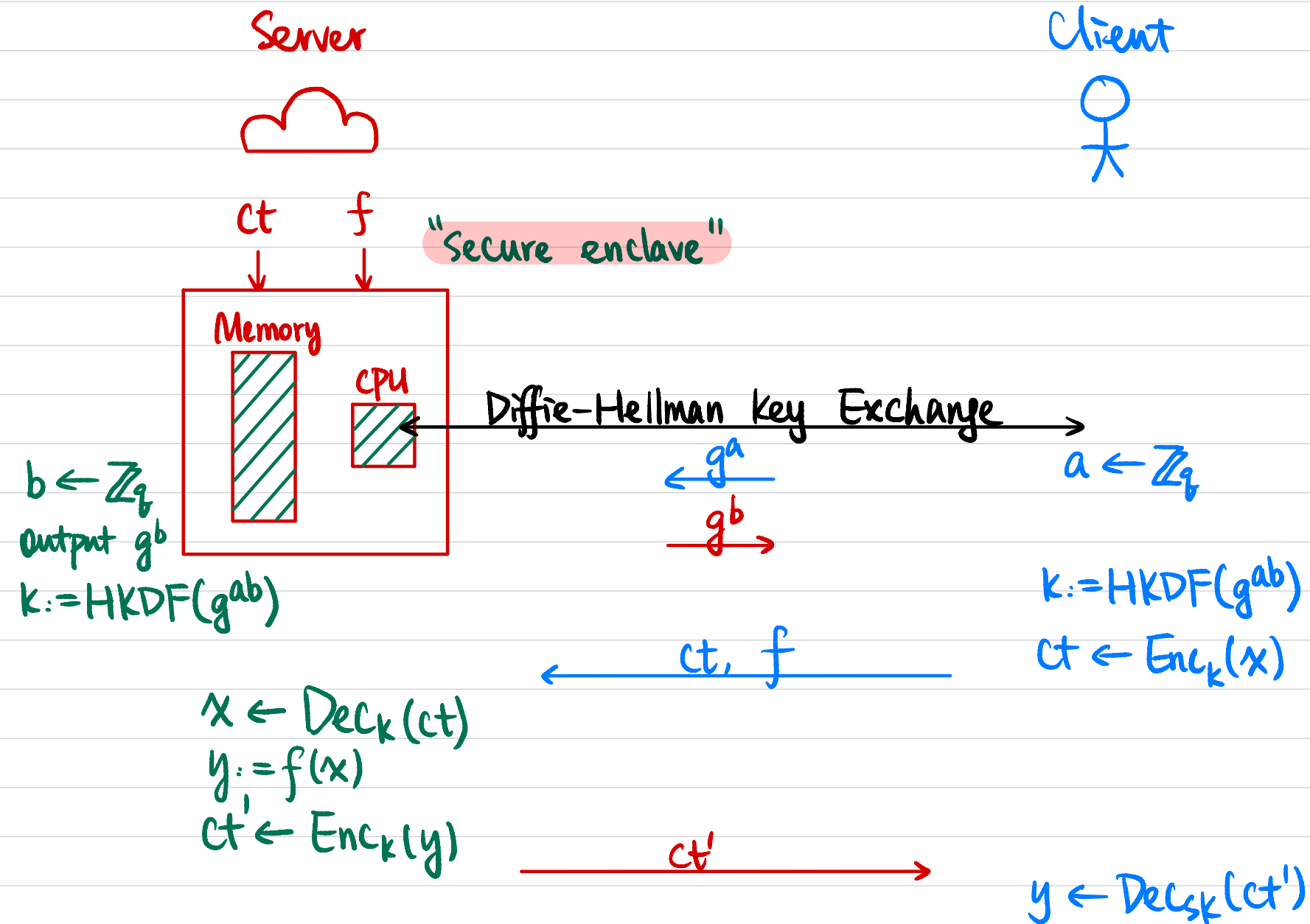
$\xrightarrow{ct'}$

$f(x) \leftarrow \text{Dec}_{sk}(ct')$

Outsourcing Computation by Secure Hardware

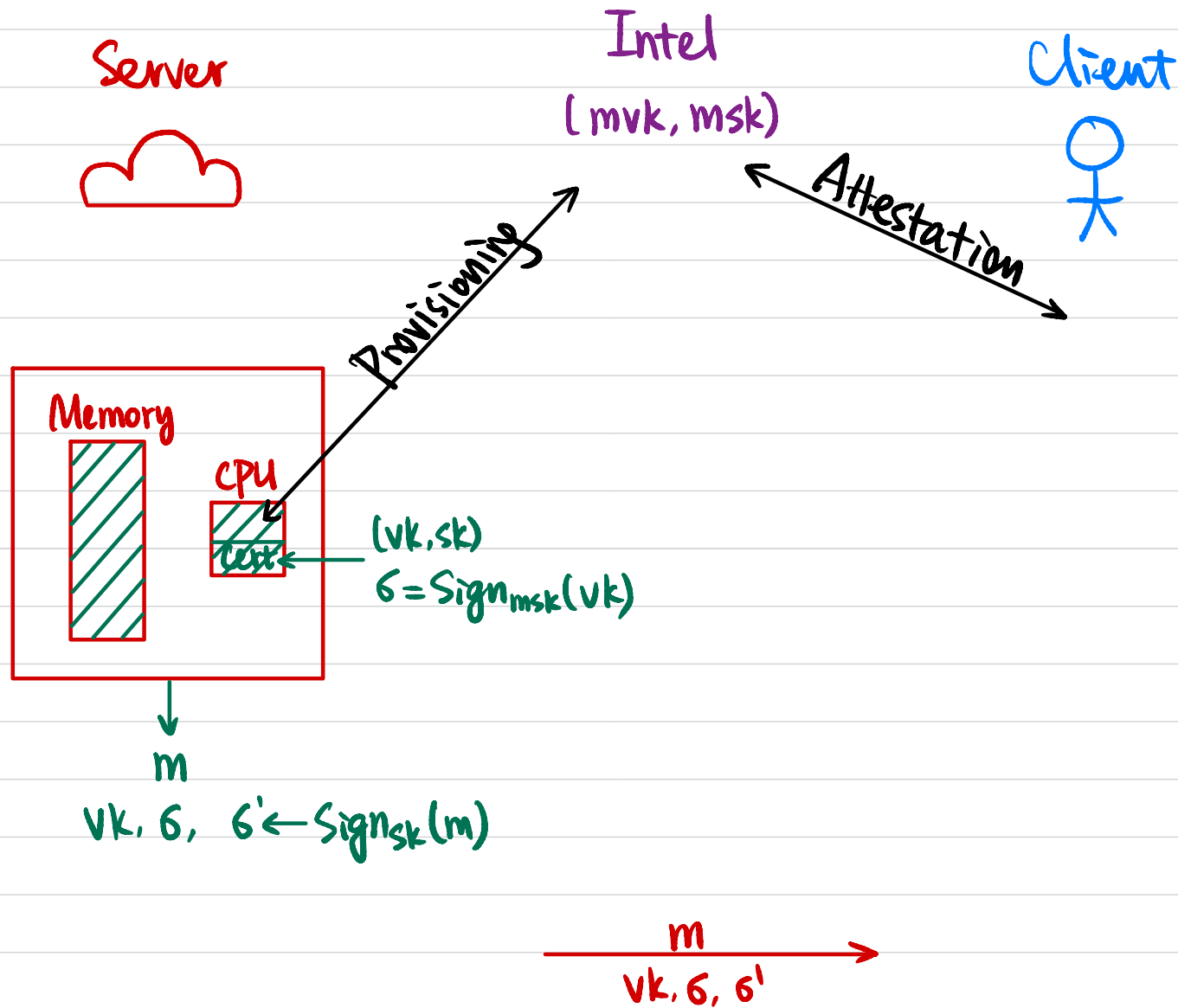


Intel Software Guard Extension (SGX)



What could go wrong?

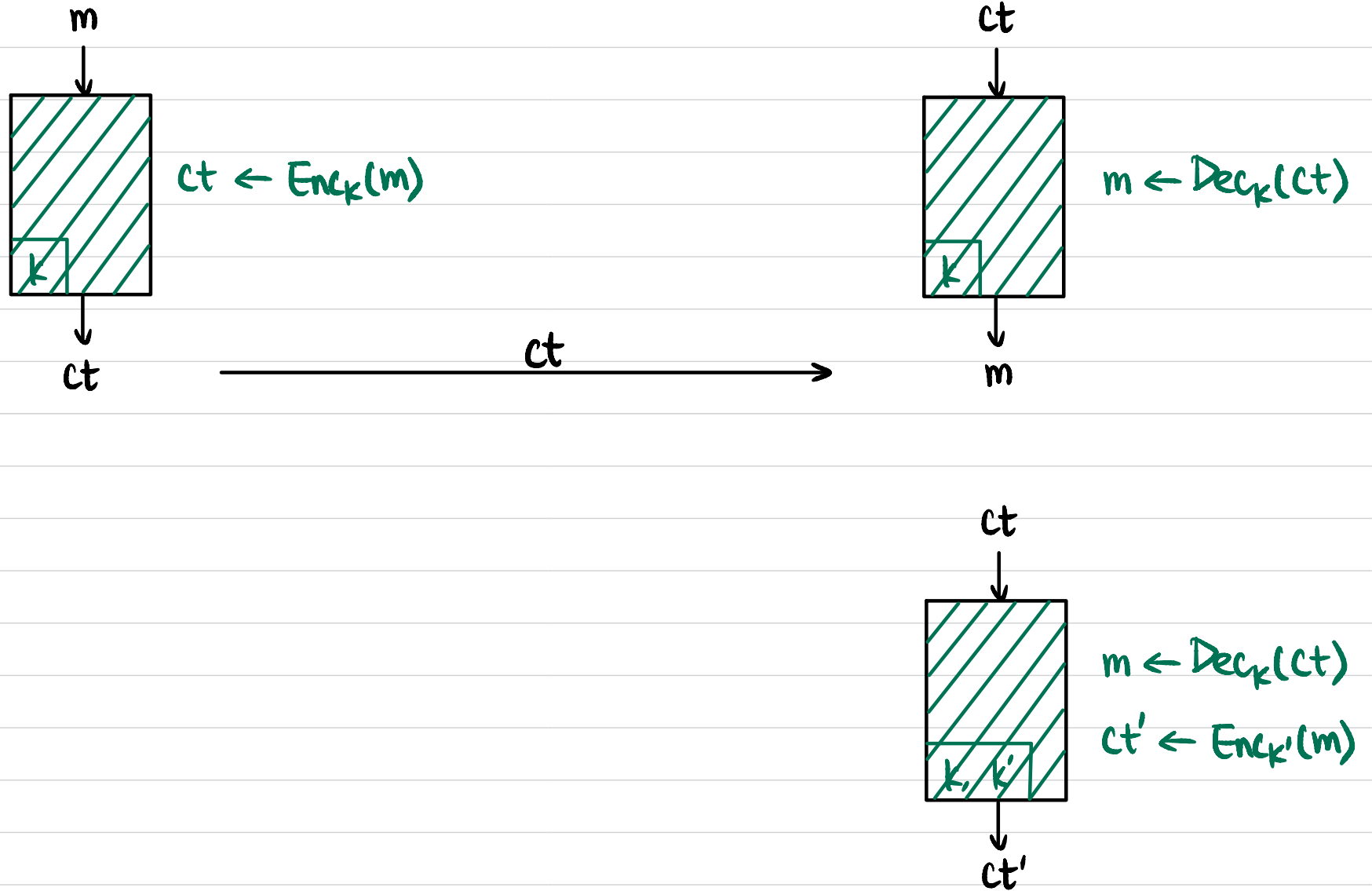
Intel Software Guard Extension (SGX)



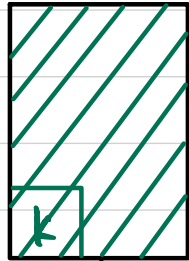
Constraints & Attacks

- Trust in hardware
- Trust in Intel
- Limited memory size: 128 MB
- Replay attacks
- Side-channel attacks : memory access pattern
 - ↳ fix: Oblivious RAM (ORAM)
 - overhead $\Theta(\log N)$

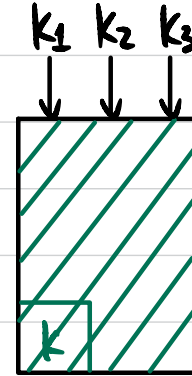
Hardware Secure Module (HSM)



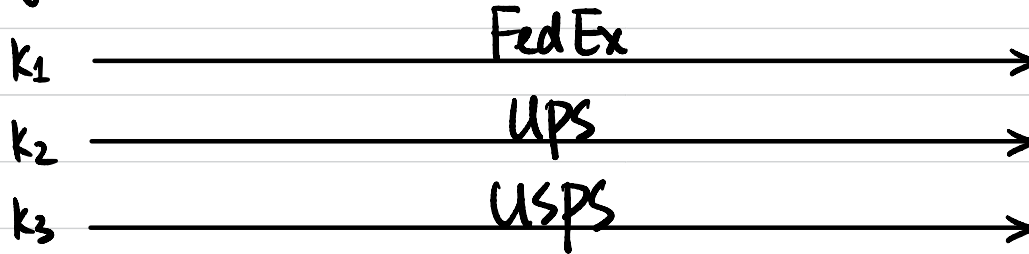
Key Agreement



Sample k_1, k_2, k_3 s.t.
 $k_1 \oplus k_2 \oplus k_3 = K$



$K := k_1 \oplus k_2 \oplus k_3$



Transactions in Real Life

Alice



Alice → Starbucks \$3

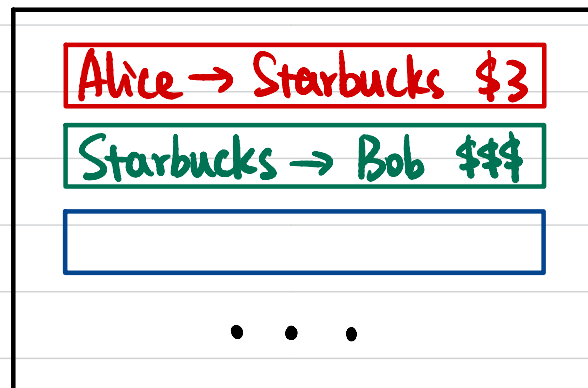
Starbucks → Bob \$\$\$

Bank of America

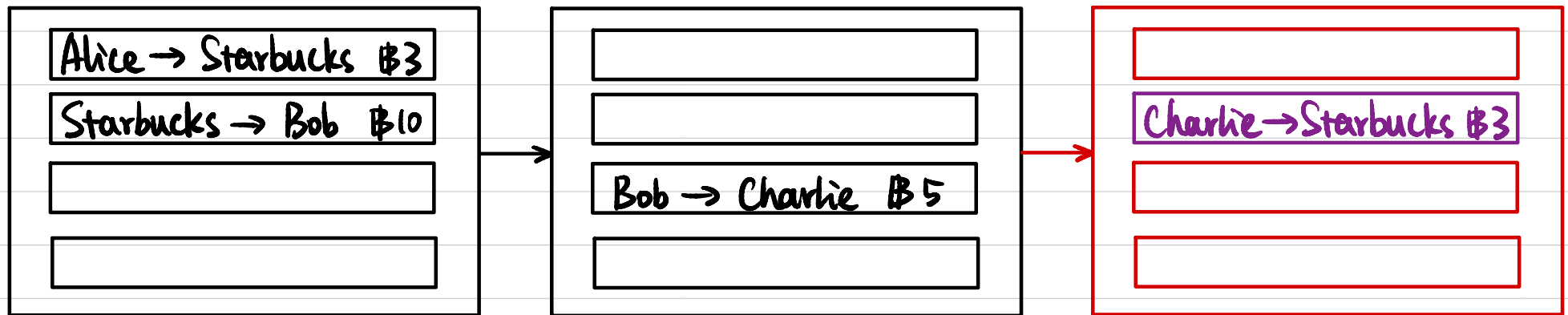


- ① initiated by sender
- ② enough balance in sender's account

A trusted party that maintains a private ledger



Blockchain



- **Public** ledger that everyone can view & verify
- Maintained by "miners" in a **distributed** way

Step 1: Charlie wants to make a transaction Charlie → Starbucks \$3
↳ broadcasts it to the entire network

Step 2: All miners collect all transactions in the network

- Verify validity $\left\{ \begin{array}{l} \textcircled{1} \text{initiated by sender} \leftarrow \text{How?} \\ \textcircled{2} \text{enough balance in sender's account} \end{array} \right.$
- Agree on next block $\leftarrow \text{How?}$

Step 3: Repeat

Transaction Authentication

Alice: $(VK_A, SK_A) \leftarrow \text{KeyGen}(1^\lambda)$

Bob: $(VK_B, SK_B) \leftarrow \text{KeyGen}(1^\lambda)$

Charlie: $(VK_C, SK_C) \leftarrow \text{KeyGen}(1^\lambda)$

Starbucks: $(VK_S, SK_S) \leftarrow \text{KeyGen}(1^\lambda)$

Bob $\rightarrow$ Charlie \$5 :

$$m_1 = (VK_B, VK_C, 5) \quad \sigma_1 \leftarrow \text{Sign}_{SK_B}(m_1)$$

Charlie $\rightarrow$ Starbucks \$3 :

$$m_2 = (VK_C, VK_S, 3) \quad \sigma_2 \leftarrow \text{Sign}_{SK_C}(m_2)$$

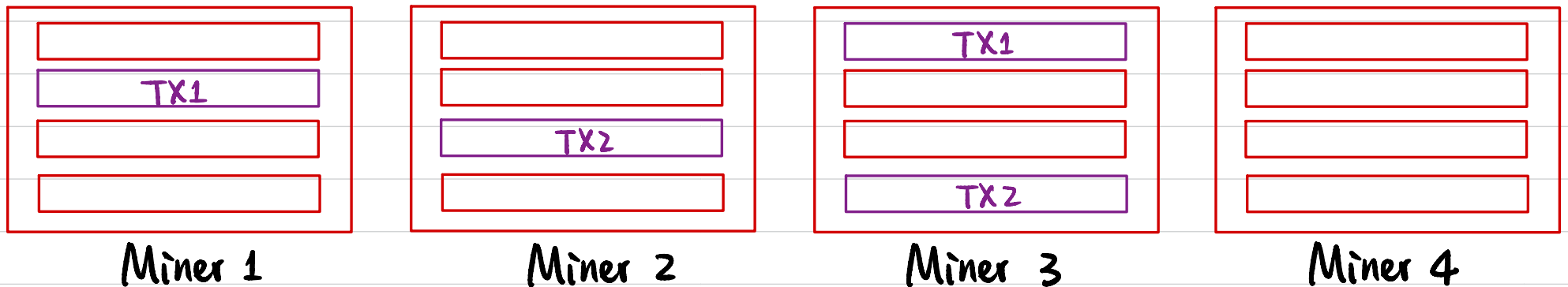
Consensus Protocol

TX1 = Charlie → Starbucks \$3 :

$$m_2 = (vk_c, vk_s, 3) \quad \sigma_2 \leftarrow \text{Sign}_{sk_c}(m_2)$$

TX2 = Charlie → Alice \$4 :

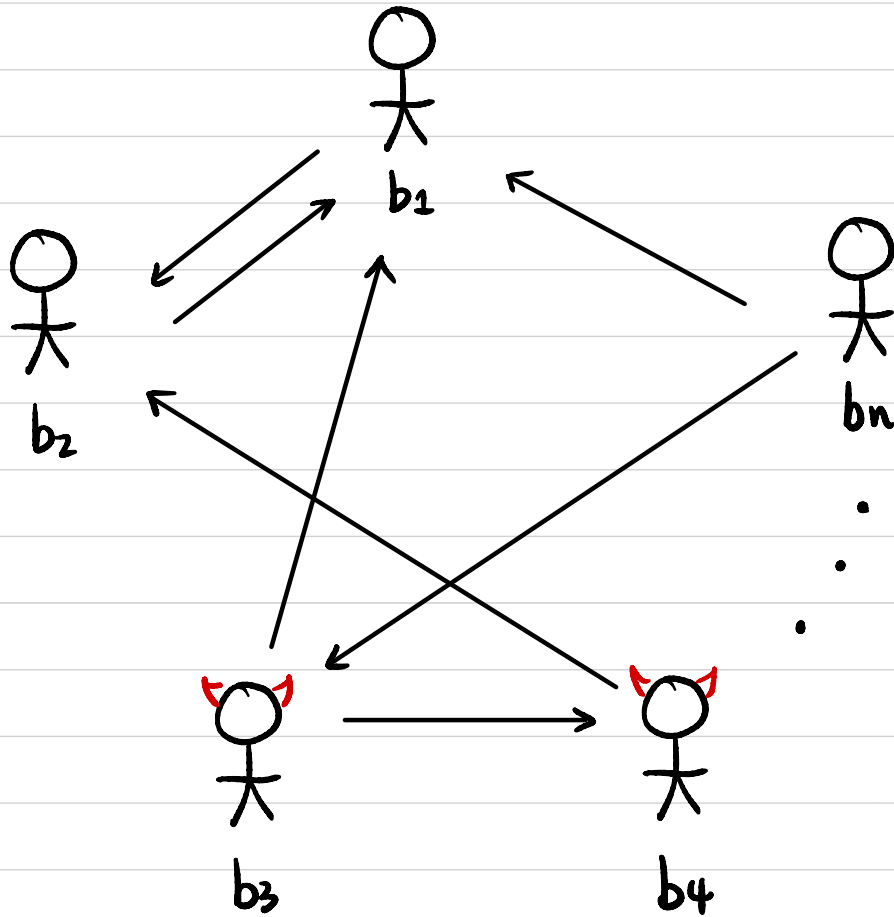
$$m_3 = (vk_c, vk_a, 4) \quad \sigma_3 \leftarrow \text{Sign}_{sk_c}(m_3)$$



WANT: ① All miners agree on the same block

② New block is valid

Byzantine Agreement



Agree on a block

(Guaranteed Output Delivery)

Byzantine Fault Tolerance (BFT) Protocol:

If $n \geq 3t + 1$,^{necessary}

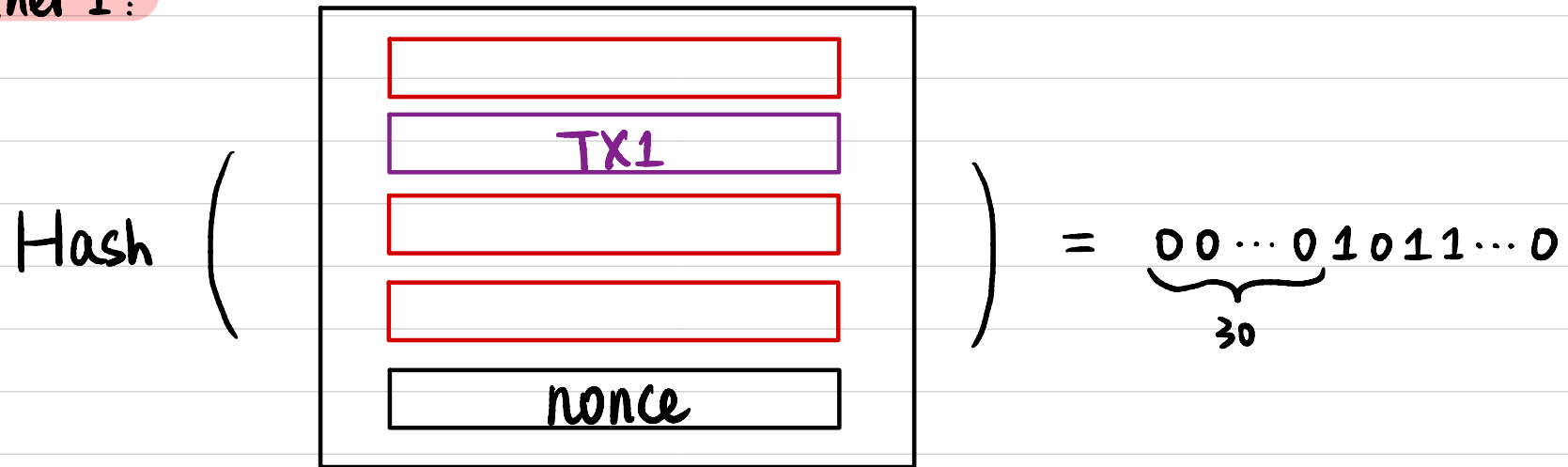
then it's possible to reach consensus.

Assume $t < n/3$, then agree on a valid block.

Any problem?

Proof of Work (PoW)

Miner 1:

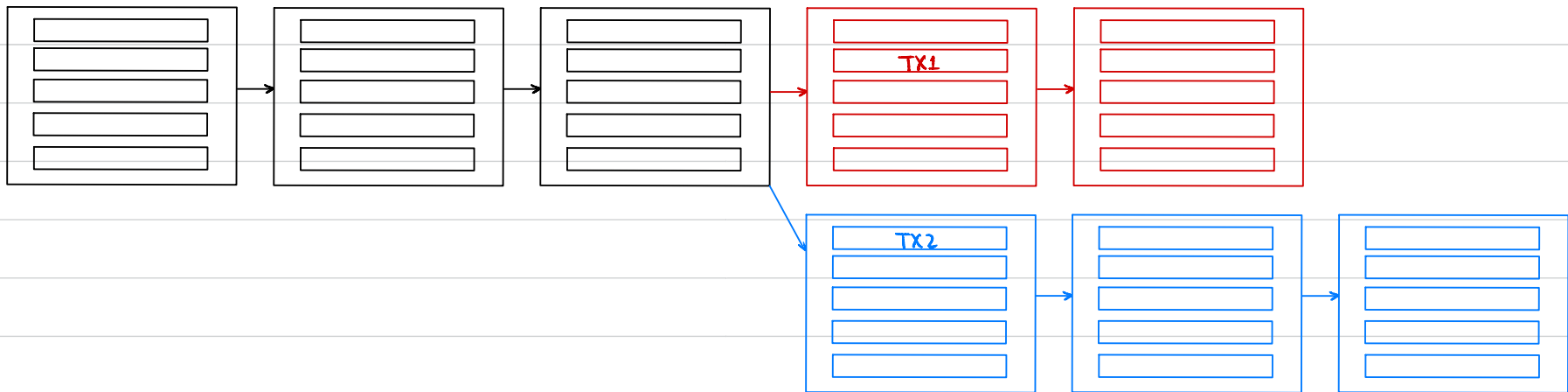
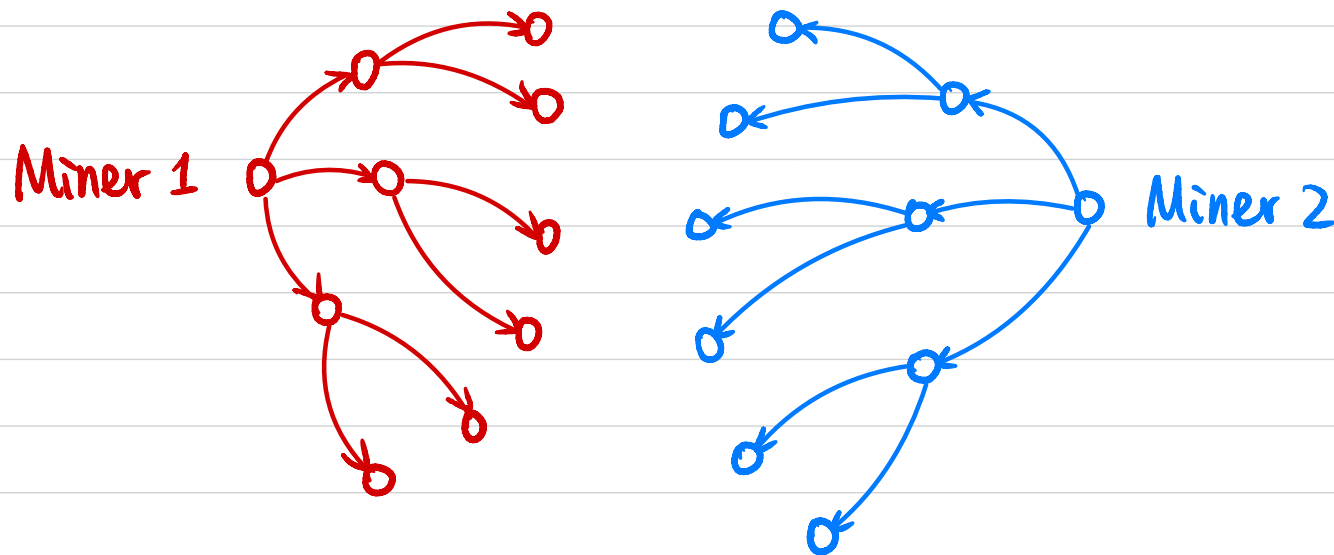


Find nonce s.t. Hash(block) has ≥ 30 leading 0's.

Consensus Protocol:

Whoever first finds a block that hashes to a value w/ ≥ 30 leading 0's, that block becomes the next block.

Proof of Work (PoW)



Longest Chain Rule: Always adopt the longest chain.

Assuming **honest majority of computation power**, the longest chain is always valid.

Blockchain

- Settlement of a transaction:

Included in a block which is ≥ 6 blocks deep (~ 1 hr)

- Dynamically adjust # leading 0's s.t. each block takes ~ 10 min to mine

Last 1 hr: > 6 blocks: increase # leading 0's

< 6 blocks: decrease # leading 0's

- Miners' motivation:

- transaction fee

- new coin generated in each block goes to miner

- Extensions

- Proof of Stake (PoS)

- Anonymous transactions (zk-SNARKs)

- Smart Contracts

- Public ledger