This Lecture:

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

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

\( \lambda: \) 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)**

\[ x_1 = \begin{array}{c} 1 \end{array} \overrightarrow{0110101011} \quad x_2 = \begin{array}{c} 0 \end{array} \overrightarrow{001101011} \]

Design Principle: “Avalanche Effect”

A one-bit change in the input should “affect” every bit of the output.
Substitution-Permutation Network (SPN)

Step 1: Key Mixing
\[ \chi := \chi \oplus k \]

Step 2: Substitution (Confusion Step)
\[ S_i: \{0,1\}^8 \rightarrow \{0,1\}^8 \] (S-box)

- Public permutation/one-to-one map
- 1-bit change of input
- \rightarrow 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

A single round of SPN

“Confusion-Diffusion Paradigm”
Substitution-Permutation Network (SPN)

An SPN is invertible given the master key.

How to compute $F_{k_1}(y)$?

Why do we need a final key mixing step?
Feistel Network

\[ x = \begin{array}{c|c}
L_0 & f_k_1 \\
\hline & \\
R_0 & f_1 \\
\hline & \\
L_1 & f_k_2 \\
\hline & \\
R_1 & f_2 \\
\hline & \\
L_2 & f_k_3 \\
\hline & \\
R_2 & L_3 \\
\hline & \\
\end{array} \]

\[ y = \begin{array}{c|c}
L_3 & \text{round function} \\
\hline & \\
R_3 & f_k(y) \text{?} \\
\hline & \\
\end{array} \]

3-round Feistel Network

\[ f_{k_i}: \{0, 1\}^{n/2} \rightarrow \{0, 1\}^{n/2} \]

How to compute \( f_k(y) \) ?

Attacks on reduced-round Feistel Network
Data Encryption Standard (DES)

16-round Feistel Network

DES mangler function

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

block length $n=64$

master key length $\lambda = 56$

Key Schedule:

Key: 28-bit 28-bit (master key)

$E$: expansion function

$E: \{0, 1\}^m \rightarrow \{0, 1\}^m$

A: 16-bit B: 16-bit E: 16-bit 16-bit 16-bit
Data Encryption Standard (DES)

**DES mangler function**

- **S-box:** \( \{0,1\}^6 \to \{0,1\}^4 \)
  - 1) "4-to-1":
    - Exactly 4 inputs map to same output
  - 2) 1-bit change of input
    - \( \to \) at least 2-bit change of output

**Key mixing**

- 48-bit intermediate
- 48-bit sub-key

**Substitution**

- \( S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8 \)

**Permutation**

- 4-bit from each S-box will affect the input to 6 S-boxes in the next round
Outsourcing Computation by FHE

Client

Data \( x \)

Key \( sk \)

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

\( ct, f \)

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

\( ct' \)

\( f(x) \leftarrow \text{Dec}_sk(ct') \)
Outsourcing Computation by Secure Hardware

Server

\[ \text{ct} \xrightarrow{f} \text{ct}' \]

\[ \text{ct}' \xrightarrow{\text{ct, } f} \]

\[ x \leftarrow \text{Dec}_sk(\text{ct}) \]

\[ y \leftarrow f(x) \]

\[ \text{ct}' \leftarrow \text{Enc}_sk(y) \]

Client

Data \( x \)

Key \( sk \)

\[ \text{ct} \leftarrow \text{Enc}(x) \]

\[ y \leftarrow \text{Dec}_sk(\text{ct}') \]
Intel Software Guard Extension (SGX)

What could go wrong?
Intel Software Guard Extension (SGX)

Server

Intel
(mvk, msk)

Client

Attestation

Memory

CPU

Provisioning

$\text{Signal}_{\text{msk}}(\text{vk})$

$\text{Vk}, \text{b}, \text{b}' \leftarrow \text{Sign}_{\text{sk}}(m)$

$m$

$\text{Vk}, \text{b}, \text{b}'$
Constraints & Attacks

- Trust in hardware
- Trust in Intel
- Limited memory size: 128 MB
- Replay attacks
- Side-channel attacks: memory access pattern

\[ \text{fix: Oblivious RAM (ORAM)} \]

\[ \text{overhead } \Theta(\log N) \]

\[ \text{memory size} \]
Hardware Secure Module (HSM)

\[ ct \leftarrow \text{Enc}_k(m) \]

\[ m \leftarrow \text{Dec}_k(ct) \]
Key Agreement

Sample $k_1, k_2, k_3$ s.t.

$$k_1 \oplus k_2 \oplus k_3 = k$$

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

$k_1 \xrightarrow{FedEx} \quad k_2 \xrightarrow{UPS} \quad k_3 \xrightarrow{USPS}$
Transactions in Real Life

Alice -> Starbucks $3

Starbucks -> Bob $$$

A trusted party that maintains a private ledger

1. initiated by sender
2. enough balance in sender's account
Blockchain

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

Step 1: Charlie wants to make a transaction
- broadcasts it to the entire network

Step 2: All miners collect all transactions in the network
- Verify validity
  - initiated by sender
  - enough balance in sender's account
- Agree on next block

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 B5:

\[ m_1 = (VK_B, VK_C, 5) \quad g_2 \leftarrow \text{Sign}_{SK_B}(m_1) \]

Charlie \rightarrow Starbucks B3:

\[ m_2 = (VK_C, VK_S, 3) \quad g_2 \leftarrow \text{Sign}_{SK_C}(m_2) \]
Consensus Protocol

\[ \text{TX1} = \text{Charlie} \rightarrow \text{Starbucks} B3 : \]

\[ m_2 = (\text{VK}_c, \text{VK}_s, 3) \quad s_2 \leftarrow \text{Sign}_{SK_c}(m_2) \]

\[ \text{TX2} = \text{Charlie} \rightarrow \text{Alice} B4 : \]

\[ m_3 = (\text{VK}_c, \text{VK}_a, 4) \quad s_3 \leftarrow \text{Sign}_{SK_c}(m_3) \]

**WANT:**

1. All miners agree on the same block
2. New block is valid
Byzantine Agreement

Byzantine Fault Tolerance (BFT) Protocol:

If $n \geq 3t+1$, then it's possible to reach consensus.

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

Any problem?

Agree on a block

(Guaranteed Output Delivery)

Sybil Attack
Proof of Work (PoW)

Miner 1:

\[
\text{Hash}\left(\begin{array}{c}
\text{TX1} \\
\text{nonce}
\end{array}\right) = 00...01011...0_{30}
\]

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

Consensus Protocol:

Whoever first finds a block that hashes to a value w/ \( \geq 30 \) 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 $\geq 6$ blocks deep ($\sim 1$ hr)

- Dynamically adjust # leading 0's s.t. each block takes $\sim 10$ 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-SNARGs)
  - Smart contracts
  - Public ledger