New Kid on the Block
Practical Construction of Block Ciphers

Foundations of Cryptography
Computer Science Department
Wellesley College

Fall 2016

Table of contents

Introduction

Substitution-permutation networks

Feistel Networks

The Data Encryption Standard
Block ciphers

- Recall that block cipher is an efficient, keyed permutation $F : \{0, 1\}^n \times \{0, 1\}^\ell \to \{0, 1\}^\ell$.
- That is, $F_k(x) \overset{\text{def}}{=} F(k, x)$ is a bijection and both it and its inversion $F_k^{-1}$ are efficiently computable for all $k$.
- Here $n$ is the key length and $\ell$ is block length.

*Both the key and block lengths are constant, so we are living in the world of concrete rather than asymptotic security.

Security requirements for block ciphers

**Remark.** Concrete security requirements are quite stringent. A block cipher is generally considered “good” if the best known attack (without preprocessing) has time complexity equivalent to a brute-force attack for the key. Recall

**Definition 3.28.** Let $F : \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^*$ an efficient keyed permutation. We say that $F$ is a strong pseudorandom permutation if for all probabilistic polynomial-time distinguishers $D$, there exists a negligible function $\text{negl}$ such that:

$$\left| \Pr[D^{F_k(\cdot), F_k^{-1}(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot), f^{-1}(\cdot)}(1^n) = 1] \right| \leq \text{negl}(n),$$

where $k \leftarrow \{0, 1\}^n$ is chosen uniformly at random and $f$ is chosen uniformly at random from the set of permutations mapping $n$-bit strings to $n$-bit strings.

**Remark.** Block ciphers are designed (at the very least) to behave as (strong) pseudorandom permutations.
Twin goals of modern encryption

- **Confusion**: Make the statical relationship between the ciphertext and the key value as complex as possible.
- **Diffusion**: Dissipate the statistical structure of the plaintext throughout the ciphertext.

- A **Substitution-Permutation Network SPN** attempts both; substitution for confusion and permutation for diffusion.

*Both ideas and promotion of substitution-permutation networks are due to Claude Shannon.*

A single round of a substitution-permutation network

- We fix a public “substitution function” $S$ called an **S-box**, and let the key $k$ define the function $f(x) = S(k \oplus x)$.
- For example, suppose SPN has a 64-bit clock length based on a collection of 8-bit S-boxes, $S_1, \ldots, S_8$:
  1. **Key mixing**: Set $x := x \oplus k$.
  2. **Substitution**: Set $x := S_1(x_1) \parallel \ldots \parallel S_8(x_8)$, where $x_i$ is the $i$th byte of $x$.
  3. **Permutation**: Permute the bits of $x$ to obtain the output of round.
The full substitution-permutation network

- The output of each round is fed as input to the next round.
- After the last round there is a final key-mixing step \((x := x \oplus k)\).*
- Different round keys are used in each round which are derived from the actual or master key according to a key schedule.

*Since we assume the S-boxes and the mixing permutations are public, without this final key-mixing the last substitution and permutations steps would be useless.

Substitution-permutation networks are invertible

**Proposition 6.3** Let \( F \) be a keyed function defined by an SPN in which the S-boxes are all permutations. Then regardless of the key schedule and number of rounds, \( F_k \) is a permutation for any \( k \).

**Proof.** We show that given the output of the SPN and the key, it is possible to invert any single round. The mixing permutation is clearly invertible and all the S-box are permutations, so these too can be inverted. The result is then XORed with the appropriate sub-key to obtain the input. 

\[ \square \]
**The avalanche effect**

The avalanche effect: A small change in either the plaintext or the key produces a significant change in the ciphertext.

One way to induce the avalanche effect in a SPN is to ensure:

1. The S-boxes are designed so that changing a single input bit changes at least **two bits** in the output.
2. The mixing permutations are designed so that the output bits an any S-box are used as input to **multiple** S-boxes in the next round.

<table>
<thead>
<tr>
<th>Round</th>
<th>Number of bits that differ</th>
<th>Number of bits that differ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>39</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>31</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>29</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>42</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>44</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>32</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>26</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>29</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>34</td>
<td>16</td>
</tr>
</tbody>
</table>

(a) Two plaintexts differ by one bit (all zeros and a one followed by 63 zeros.)
(b) The two keys also differ by just one bit.

**How this works in practice**

- Suppose 8-bit S-boxes and the mixing permutations are chosen as above for a 128-bit block size SPN.
- Now consider two inputs that differ by a single bit.
• In the early 1970s, Horst Feistel developed a sequence of fixed transpositions and key-dependent, multipartite non-linear substitutions (*LUCIFER*) that produced a thorough amalgamation.

• In this simplified illustration of LUCIFER we see a plain text input of a single 1 and fourteen 0s transformed by the non-linear S-boxes into an avalanche of eleven 1s.

• Feistel was successful enough to upset the NSA.

**Attacking a one-round substitution-permutation network**

• Suppose the SPN $F$ consists of a single full round and no final key-mixing.

• An adversary can easily learn the secret key $k$ given only a single input/output pair $(x, y = F_k(y))$. How?
Attacking a one-round substitution-permutation network with key-mixing

The attacker fixes a pair, \((x, y)\) could simple try all 64-bit possibilities for second-round mixing key, \(k_2\), then use above attack to construct \(2^{64}\) candidate keys \(k_1 \| k_2\). Additional pairs narrow the field. But the attacker can do better:

1. First enumerate over all possible first bytes of \(k_2\).
2. Then XOR each with of these with the first byte of \(y\) to obtain candidates for the outputs of the first \(S\)-box.
3. Backing each of these up through the first \(S\)-box, the adversary has 256 candidates for the the first bit of \(x \oplus k_1\) and hence 256 candidates for the first byte of \(k_1\).

Feistel: An alternative to substitution-permutation networks

- A Feistel network gives a way to construct invertible functions from non-invertible components.
- The goal is a block cipher that has an “unstructured” behavior*. Requiring all components to be invertible inherently introduces structure.
- Like SPNs, Feistel networks operate in a series of rounds each with a keyed round function constructed from \(S\)-boxes and permutations.

*So it looks random.
Feistel round functions

- The $i$th round function, $\hat{f}_i$, takes as input a sub-key $k_i$ and an $\ell/2$-bit string and outputs an $\ell/2$-bit string.

- A master key $k$ determines a series of round keys $k_i$. The round function $f_i : \{0,1\}^{\ell/2} \rightarrow \{0,1\}^{\ell/2}$ is defined by $f_i(R) \overset{\text{def}}{=} \hat{f}_i(k_i, R)$.

- The output of the $i$th round $(L_i, R_i)$ is

  $$L_i := R_{i-1} \text{ and } R_i := L_{i-1} \oplus f_i(R_{i-1}).$$

*So it looks random.

Feistel networks are invertible

**Proposition 6.4.** Let $F$ be a keyed function defined by a Feistel network. Then regardless of the round functions $\{\hat{f}_i\}$ and the number of rounds, $F_k$ is an efficiently invertible permutation for all $k$.

**Proof.** We need only show that each round is invertible if the $\{f_i\}$ are known.

Given $(L_i, R_i)$, we first compute $R_{i-1} := L_i$. Then compute

$$L_{i-1} = R_i \oplus f_i R_i - 1).$$
The return of LUCIFER

- LUCIFER was redesigned as a 16-round Feistel network on 128-bit blocks and 128-bit keys and was submitted by IBM as a candidate for the Data Encryption Standard.

- It became DES after the National Security Agency reduced its block size to 64 bits and its key size to 56 bits* and was adopted as the Federal Information Processing Standard for the US in 1977.

- Believe it or not, it remains in wide use today in its triple-DES incarnation.

*Which means that it is now vulnerable to brute-force attacks.

The Data Encryption Standard

- DES is a 16-round Feistel network with a block length of 64 bits and a key length of 56 bits.

- The same function, \( \hat{f} \) called the DES mangler function, is used for all 16 rounds.

- It takes a 48-bit round key derived from the 56-bit master using a relatively simple key schedule.

- We give a high-level overview of the main components of DES. A homework exercise is available for those interested in more details.
The DES mangler function

- Computation of \( \hat{f}(k_i, R) \) with \( k_i \in \{0, 1\}^{48} \) and \( R \in \{0, 1\}^{32} \) begins by duplicating half the bits of \( R \) to obtain \( R' := E(R) \), \( E \) the \textit{expansion function}.

- The expanded \( R' \) is XORed with \( k \) and the resulting value is divided into 8 6-bit-blocks. Each block is passed through a different \( S \)-box.*

- Finally a mixing permutation is applied to the bits and it off to the next round.

*The \( S \)-boxes are publicly known, but unlike our previous SPNs, they are not invertible.

*The eight \( S \)-boxes at the core of the mangler function were very carefully designed. Even slight modification would make DES more vulnerable to attack.*

*The Feistel version of LUCIFER was susceptible to differential cryptanalysis; for about half the keys, the cipher could be broken with 236 chosen plaintexts and 236 time complexity. This is one of the reasons NSA redesigned them.
**Time to worry: Security of DES**

- After 30 years of intensive study, the best known practical attack on DES is exhaustive search.* However, a 56-bit key isn’t very big.

<table>
<thead>
<tr>
<th>Key Size (bits)</th>
<th>Number of Alternative Keys</th>
<th>Time required at 1 encryption/μs</th>
<th>Time required at 10^9 encryptions/μs</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>2^32 = 4.3 × 10^9</td>
<td>2^31 μs = 35.8 minutes</td>
<td>2.15 milliseconds</td>
</tr>
<tr>
<td>56</td>
<td>2^56 = 7.2 × 10^16</td>
<td>2^5 μs = 1142 years</td>
<td>10.01 hours</td>
</tr>
<tr>
<td>128</td>
<td>2^128 = 3.4 × 10^38</td>
<td>2^12 μs = 5.4 × 10^7 years</td>
<td>5.4 × 10^9 years</td>
</tr>
<tr>
<td>26 characters</td>
<td>2^6 = 4 × 10^26</td>
<td>2 × 10^10 μs = 6.4 × 10^2 years</td>
<td>6.4 × 10^9 years</td>
</tr>
</tbody>
</table>

- A second concern is the relatively short block length of DES. The proof of security for CTR mode (Theorem 3.32) show that even when a completely random function is used an attacker can break it with probability 2q^2/2^ℓ if it obtains q plaintext/ciphertext pairs.

*In your homework you get the chance to investigate some theoretical attack, but these require a large number of input/output pairs which would be difficult to obtain.

**Double encryption**

- Let $F$ be a block cipher with an $n$-bit key length and $ℓ$-bit block length. Define a new block cipher with a key length of $2n$ by

$$F'_{k_1,k_2}(x) \overset{\text{def}}{=} F_{k_2}(F_{k_1}(x)).$$

- For the case where $F$ is DES, we obtain a cipher $F'$ call 2DES that takes a 112-bit key. Exhaustive search is now out of reach.
Meet in the middle attack

Say adversary is given a single input/output pair \((x, y)\), where 
\[ y = F_{k_1^*}^{\dagger}(x) = F_{k_2^*}^{\dagger}(F_{k_1^*}(x)) \]
for unknown \(k_1^*, k_2^*\).

1. For each \(k_1 \in \{0, 1\}^n\) compute 
\[ z := F_{k_1}(x) \]
and store \((z, k_1)\) in list \(L\).

2. For each \(k_2 \in \{0, 1\}^n\) compute 
\[ z := F_{k_2}^{-1}(x) \]
and store \((z, k_2)\) in list \(L'\).

3. Entries \((z_1, k_1) \in L\) and 
\((z_2, k_2) \in L'\) are a match if \(z_1 = z_2\).

The attack takes \(O(n \cdot 2^n)\) time and requires space \(O((n + \ell) \cdot 2^n)\) space and finds pairs \((k_1, k_2)\) with 
\[ F_{k_1}(x) = F_{k_2}^{-1}(y). \]

Triple DES with two keys

- An obvious counter to the meet-in-the-middle attack is to use three stages of encryption with three different keys.

- As an alternative, Tuchman proposed a triple encryption using only two keys 
\[ y = F_{k_1}(F_{k_2}^{-1}(F_{k_1}(x))). \]

- Triple-DES’s relatively small block length and the fact that is slow since it requires 3 full block-cipher operations, led to its replacement by the Advanced Encryption Standard.