"Theoretical" example: Single-round substitution-permutation network

\[ \text{Known plaintext: single pair } (x, y = F_k(x)) \]
\[ \text{Known, deterministic } y' = x \oplus y' = k \]
\[ \Rightarrow \text{key recovery from single input/output pair} \]

Two-round substitution-permutation network

\[ \text{Known plaintext: } (x_1, y_1 = F_k(x_1)), (x_2, y_2 = F_k(x_2)) \]

\[ \text{Attack: Do } 2^{20} \text{ work to get } 2^{4} \text{ possible } (k_1, k_2) \text{ pairs} \]

\[ \text{Use } (x_1, y_1) \text{ pair to recover: } 2^{20} \text{ work } \approx \left( \frac{p}{f(x)} \right) \times 2^4 \]
\[ \Rightarrow 8 \text{ pairs; probably only 1 pair left, work: } 8 \times 2^{20} = 2^{23} \]

Repeat for 16 rounds: \[ k_2 = 7 \]
\[ 16 \times 2^{20} = 2^{27} < 2^{128} \]

\[ \Rightarrow \text{known permutation} \]
\[ \Rightarrow \text{value of } x, \text{ influenced by } \leq 9 \text{ S-boxes} \]
\[ \Rightarrow \leq 16 \text{ bits of } k_1, \text{ influence value} \]

\[ \text{Considered} \]
\[ \text{Observation} \]
\[ \Rightarrow \text{deterministic, known} \]
\[ \Rightarrow \text{deterministic, known} \]
\[ \Rightarrow \text{deterministic, known} \]

\[ \Rightarrow \text{not enough difference creates} \]
{
\text{Attacking 2DES}\\
F_{k_1, k_2}^1(x) = F_{k_2}(F_{k_1}(x))\\
\text{DES \Rightarrow } k_1, k_2 = 112 \text{ bits } = O(1) \text{ from exhaustive search}\\
\text{Meet-in-the-middle attack}\\
(x, y = F'_{k_1, k_2}(x))\\
\begin{align*}
\text{1. For each } k_1 \text{ of } 2^{112}, \text{ compute } z = F_{k_1}(x) \text{ and store } (z, k_1) \\
\text{2. For each } k_2 \text{ of } 2^{112}, \text{ compute } z = F_{k_2}^{-1}(y) \text{ and store } (z, k_2) \\
\text{3. Find matches } (z_i, x_i), (z_j, y_j), (z_i = z_j) \Rightarrow (x_i, y_j) \text{ is possible key.}
\end{align*}
\begin{align*}
F_{k_1}(x) & \rightarrow z_1 \\
F_{k_2}(x) & \rightarrow z_2 \\
F_{k_2}^{-1}(y) & \rightarrow y
\end{align*}
\text{Running Time:}\\
\begin{align*}
\text{step 1: } & 2^{112} \text{ time} \\
\text{step 2: } & 2^{112} \text{ time} \\
\text{step 3: } & 2 \times O(n \cdot 2^{112}) \text{ time to sort, } 2^{112} \text{ time to match } \Rightarrow O(n \cdot 2^{112}) \text{ total time} \\
& O(2^{112}) \text{ space}
\end{align*}
\text{Meet-in-the-middle 3DES}\\
F_{k_1}(F_{k_2}^{-1}(F_{k_3}(x))) = F_{k_1, k_2, k_3}^1\\
\text{2}^{212} \text{ time}
\begin{align*}
F_{k_1} & \rightarrow z_1 \\
F_{k_2}^{-1} & \rightarrow z_2 \\
F_{k_3} & \rightarrow y
\end{align*}
Chosen Plaintext attack against CBC mode

CBC mode

\[
\begin{array}{c}
\text{IV} \rightarrow \oplus \rightarrow E_k \rightarrow \oplus \rightarrow E_k \rightarrow \cdots \\
\text{C}_0 \quad \text{C}_1
\end{array}
\]

Royappa 1995:
CBC mode not secure against chosen plaintext attacks: see IV or previous block to learn choose m

1. Attack observes C_1, C_2, \ldots, C_i, C'_i, \ldots, C_j.
   [attackers encrypt value c_i]

2. Attack guesses C_i, plaintext is P.

3. Attack causes victim to encrypt C_j \oplus C_{i-1} \oplus P.

4. Victim sends E_k(C_j \oplus (C_j \oplus C_{i-1} \oplus P)) = E_k(C_{i-1} \oplus P).

5. Attack compares E_k(C_{i-1} \oplus P) to C_i, matches P correct.

Dai 2002: SSH2 changes ciphertext between client & server.
SSL v3, TLS 1.0 also change between requests

Problem: 128-bit block size is a lot to brute force
Solution: In many use cases, know almost all of the key already

Problem: How does real adversary request encryptions?
Solution: Duong & Rizzo 2011

JavaScript, Java etc. allow browsers to make cross-domain requests across domains
WebSockets, HTML5
BEAST (Browser Exploit against SSL/TLS)
BEAST continued:
Threat model: Attacker can make arbitrary requests from victim chat
Attacker has reverse-access to see encrypted traffic

Ciphertext HTTPS encrypted message:
HTTP requests: known attack-controls
\[ \text{GET} /\text{index.html} \quad \text{HTTP/1.1} \]
Cookie: sessionid = ...

(bad)

want to steal

Doug+Rizzo: Pad plaintext so byte of unknown text is in a block.

i. Alice HTTPS POST \(\rightarrow\) http://bob.com/AAAAAAA ...
\[ \text{POST} /\text{AAAAAAA} \quad \text{HTTP/1.1} \]
\[ \text{Cookie: sessionid = good} \]
\[ \text{chosen by attacker} \]

2. Mallory captures cipher-text blocks C1...Cn
3. Mallory requests C1 \& CN \& P
\[ \text{brute forcing 256 chars until match} \]
4. Mallory shifts As until next block required.

Fixes:
Implement TLS 1.1 (may standard since 2006)
- Prepend empty records (fails due to bugs)
- Send one byte of data in 1st CBC-encrypted/decrypted record to randomize IV (fails due to bugs)

Adam Langley: "The internet is vast and filled with bugs."
- Use RC4 (see Lecture 3)

CRIME Attack (Compression Ratio Information Leak Made Easy)
Kelsey 2002: Compression is a side-channel if compress then encrypt and leak length

Doug+Rizzo 2012:
Attacker control: body of HTTP requests

\[ \text{POST} /\text{HTTP/1.1} \]
\[ \text{Host: bob.com} \]
\[ \text{Cookie: secret = 1234} \]

\[ \text{Cookie: secret = 0} \quad \text{body encoded/gallicue} \]

Compressed length different attacker correctly guesses byte of cookie

LZ77 Compression algorithm:
\[ \text{store output} \]
1. Start at beginning of stream
2. Repeat until done:
3. Output \((L, B, L, C)\)
   - move \(B \uparrow\) next non-match character
   - copy \(\text{last char of output} \)
4. move coding position \(L+1\) chars forward
CCA indistinguishability experiment:

\[
A \xrightarrow{\text{oracle access}} \text{Enc}_k(\cdot) \xrightarrow{\text{m}, m_0, m_1} \text{Dec}_k(\cdot) \xrightarrow{\text{c}} b^1 \xrightarrow{\text{U}} b \xrightarrow{\text{b' \neq b}} \text{Pr}[A \text{ succeeds}] \leq \frac{1}{2} + \varepsilon(n)
\]

\[k = \text{Gen}(1^n)\]
\[b \in \mathbb{Z}_2\]
\[c = \text{Enc}_k(m_b)\]

Definition: "indistinguishable encryption under a chosen ciphertext attack (CCA-secure)"

\[\text{Pr}[A \text{ succeeds}] \leq \frac{1}{2} + \varepsilon(n)\]