Timing Attacks
(Kocher '96)

Modular exponentiation: \( m = c^d \mod n \)

\[ m = 1 \]

\[ \text{for } i = 0 \ldots d-1 \text{ (d has k bits)} \]

\[ \text{if } d[i] = 1: \]

\[ m = c \cdot m \mod n \]

\[ m = m^2 \mod n \quad \text{might sometimes be slow (repeated squaring)} \]

Hypothetically, running time depends on Hamming weight of \( d \Rightarrow \text{Bad} \)

How to turn into full attack?

Assumption:
Time taken by \( m_i = c \cdot m \mod n \) depends on \( m_i \)

\[ \Rightarrow \text{if guess } d[i] \text{ wrong, no correlation} \]

We have access to the implementation and can check our work.

Attack:
1. Choose \( c_1, \ldots, c_k \) and measure time of decryption \( T_j = \sum_{i=0}^{d} t_{ij} + e_j \)
2. For bit \( b = 0 \ldots k \):
   - Guess value of \( d[0\ldots b] = d_i \)
   - Compute \( Z_j = T_j - \text{Time}(c_j; \text{main}) \)

\[ Z_j \approx \sum_{i=0}^{b} t_{ij} + e_j \quad \text{if correct} \]

\[ \approx \sum_{i=0}^{b} t_{ij} - (\sum_{i=0}^{b} t_{ij} + \sum_{i=b+1}^{d} s_{ij}) \quad \text{if incorrect from b} \]

3. Compute \( \text{Var}(Z_j) = \left( \sum_{i=0}^{b} \text{Var}(t_{ij}) \right) \text{ if correct} \)
\[ + \left( \sum_{i=b+1}^{d} \text{Var}(t_{ij}) \right) \quad \text{if incorrect from b} \]
Pour Analysis

Simple pour analysis
Plot pour consumption over time

Differential power analysis (DPA)
Look for statistical correlations between power traces based on intermediate computation

Block ciphers:

**DES**

**Feistel Network**

\[ \text{Plaintext} \]
\[ \rightarrow \text{Initial Permutation} \]
\[ L_0 \quad R_0 \]
\[ f \]
\[ L_1 \quad R_1 \]
\[ L_{15} \quad R_{15} \]
\[ f \]
\[ \text{Final Permutation} \]
\[ \text{Ciphertext} \]

**AES**

Key schedule

\[ \text{Key} (64 \text{bits}) \]
\[ \text{Round 1} \]
\[ \text{Round 2} \]
\[ \text{Round 3} \]
\[ \text{Mix Columns} \]

AES Rounds

\[ L \]
\[ R \]
\[ f \]
\[ L_{i-1} \]
\[ \text{Expand} \]
\[ D \]
\[ S_1 \quad S_2 \quad \ldots \quad S_8 \]
\[ \text{Permutation} \]

**DES**

\[ L_0 \quad R_0 \]
\[ f \]
\[ L_1 \quad R_1 \]
\[ L_{15} \quad R_{15} \]
\[ f \]
\[ \text{Final Permutation} \]
\[ \text{Ciphertext} \]
DPA against S-boxes

\[
P_i \xrightarrow{+2 \text{ bytes of plaintext (view as in-strand)}} \]

\[
K_i \rightarrow \oplus \rightarrow S \rightarrow Z_i
\]

**Assume:**
- Know a bunch of traces for known plaintext EP3
- Know how cipher is implemented (like is mapping of S-boxes)

**Attack:**
1. Choose a "selection function"
   - \( \text{bit} 0 \) of \( Z_i \) is 0 or 1
2. For each possible \( K_i \):
   a. Compute corresponding \( Z_i \) for each \( P_i \) in sample.
   b. Sort traces into 2 piles
      \[
      \begin{align*}
      \text{ET: bit} 0 \text{ of } Z_i = 0 \Rightarrow S_0 \\
      \text{ET: bit} 0 \text{ of } Z_i = 1 \Rightarrow S_1
      \end{align*}
      \]
   c. Compute correlation:
      \[
      D_{i,j} = \frac{\sum T_{S_0}}{1501} - \frac{\sum T_{S_1}}{151,1}
      \]
3. Pick guess for \( K_i \) with "biggest spike"
4. Repeat until all key known.
Length as a side-channel

CRIME attack
(Duong & Rizzo 2012)

Summary: compress-then-encrypt is a problem
with chosen-plaintext attack + length side-channel

Kelsey 2002: compression is a side-channel

Assumption:
Attacker can control both aT requests, append k

POST / HTTP/1.1
Host: bob.com

Cookie: secret = 12345; attacker doesn't know

Cookie: secret = 0; body < attacker can control

Insight: Compressed length differs if attacker correctly guesses byte at output

LZ77 Compression Algorithm:

1. Start at beginning of stream.
   Repeat until done:
   2. Find longest match in window for lookahead buffer
   3. Output (L, L), C
      move C, copy L chars
      move back to output
   4. Move coding position L+1 chars forward
Error messages are side-channel

Blondelacker attack (1996)

Recall: PKCS#1 padding:

\[ \begin{array}{c}
0101
\end{array} \]

Adaptive chosen-plaintext attack:

Attacker wants to find \( m = c^d \mod n \)

1. Choose integer \( s \), compute
   \[ c' = c^s \mod n \]
2. Send \( c' \) to decryption oracle.
3. Oracle responds: if \( c' \) is PKCS\#1
   
   \[ \Rightarrow ms = 0101 \ldots \]
   
   \[ \Rightarrow 2^{2(k-2)} \leq ms \mod n < 3 \left( 2^{8(k-2)} \right) \]
4. Progressively narrow range of possible \( m \) until done.

How does the oracle work in practice?

1. SSL servers returning different response if padding wrong vs. bad message
2. Timing side-channel
   \[ \Rightarrow \text{Anley forgot right} \]
Countermesures

Blinding

Compute \( m = (c \cdot r^e)^d \cdot r^{-1} \mod n \)
- must be careful not to accidentally leak \( r \)
- can be expensive

Masking

Operate on bytes before
- must update nonlinear components, like S-boxes to masked versions
- must update markings