FIT5124 Advanced Topics in Security

Lecture 7: Hacking Techniques I – Side Channel Attacks

Ron Steinfeld
Clayton School of IT
Monash University

April 2015

Hacking Techniques I

Side Channel Attacks: How to break strong cryptography using implementation 'side' information?

Implementations of secure systems can leak secret information via **side channels**. Hackers can exploit these leaks to break 'secure' systems!

Plan for this lecture: Exploitation techniques, examples, and defenses for:

- Timing side channels
- Power side channels
- Cache side channels
- Other side channels (EM, sound, ...)

Timing Side Channels

Q: How can timing the length of computations help an attacker to break a system?

A: In many implementations, time of execution leaks sensitive information!

We will look at several examples and attack techniques:

- Password verification
- RSA signature generation

Timing Side Channels: Password verification

Consider following algorithm for verifying passwords at login:

Inputs:

- $\tilde{P} = (\tilde{P}[0], \ldots, \tilde{P}[7])$: Login 8 char. password
- $P = (P[0], \ldots, P[7])$: Registered 8 char. password

Output: 'True' if $\tilde{P} = P$, 'False' otherwise.

Algorithm 4 Password verification.

Input: $\tilde{P} = (\tilde{P}[0], \ldots, \tilde{P}[7])$ (and $P = (P[0], \ldots, P[7])$)
Output: `'true'` or `'false'`

```
for j = 0 to 7 do
    if (P̃[j] ≠ P[j]) then return 'false'
end for
return 'true'
```

Q1: Is there an execution time leakage vulnerability?

Q2: How could an attacker exploit it?

Timing Side Channels: Password verification

A1: Execution time leakage vulnerability:

- 'for' loop terminates as soon as as a byte mismatch is found!
- Number of executed iterations of 'for' loop = smallest j such that $\tilde{P}[j] \neq P[j]$.

A2: Timing attack exploitation:

1. For $0 \leq n \leq 255$, the attacker proposes the 256 passwords $\widetilde{P}^{(n)} = (n, 0, 0, 0, 0, 0, 0, 0)$ and measures the corresponding running time, $\tau[n]$.
2. Next, the attacker computes the maximum running time

$$\tau[n_0] := \max_{0 \leq n \leq 255} \tau[n] \ .$$

   The correct value for the first byte of P, $P[0]$, is given by n_0.
3. Once $P[0]$ is known, the attacker reiterates the attack with

$$\widetilde{P}^{(n)} = (P[0], n, 0, 0, 0, 0, 0, 0),$$

   and so on until the whole value of P is recovered.

Timing Side Channels: RSA Signature Generation

Q: How to break a system where total execution time depends on all parts of the secret?

Example: RSA Signature Generation

Consider following 'square and multiply' algorithm for RSA 'hash and sign' signature generation:

Inputs:

- m: Message to be signed
- N: RSA signature public key modulus
- $d = (d_{k-1}, \ldots, d_0)$: RSA signature private key exponent
- μ: hash function to hash message into $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$ before signing.

Output: RSA signature $\sigma = \mu(m)^d$ mod N.

Algorithm 5 Computation of an RSA signature.

Input: $m, N, d = (d_{k-1}, \ldots, d_0)_2$, and $\mu : \{0,1\}^* \rightarrow \mathbb{Z}/N\mathbb{Z}$

Output: $S = \mu(m)^d \pmod N$

```
R_0 ← 1; R_1 ← μ(m)
for j = k − 1 downto 0 do
    R_0 ← R_0^2 (mod N)
    if (d_j = 1) then R_0 ← R_0 · R_1 (mod N)
end for
return R_0
```

Timing Side Channels: RSA Signature Generation

First execution time leakage vulnerability:

- Multiply step $R_0 \leftarrow R_0 \cdot R_1 \bmod N$ in line 4 only executed if $d_j = 1$.

But... attacker can only measure total execution time:

- Total time depends on all secret bits $d_{k-1}, \ldots, d_0$.
- Seems to reveals only number of 1s (Hamming weight) of d!

What can attacker do?

A: Look for dependence of a local computation on just one secret bit and attacker's input!

Second execution time leakage vulnerability:

- Look inside implementation of line 4 Multiply $R_0 \leftarrow R_0 \cdot R_1 \bmod N$
- Performed using efficient 'Montgomery multiplication' method.
- Montgomery method outputs the correct result but as integer y in interval $[0, 2N-1]$ (not $[0, \ldots, N-1]$).
- Hence, introduces input-dependent execution time:
  - If $y \in [N, \ldots, 2N-1]$ need to reduce mod N with a subtraction: $y \leftarrow y - N$.
  - Else, if $y \in [0, \ldots, N-1]$, don't perform subtraction.
- Time of $R_0 \leftarrow R_0 \cdot R_1 \bmod N$ in line 4 depends on R_0 and R_1 values!

Timing Side Channels: RSA Signature Generation

Timing attack exploitation idea:

- Time signature generation on many random input messages $m_1, \ldots, m_t$
- For each message m_i, inputs R_0, R_1 to line 4 Montg. Multiply for first loop iteration $j = k - 1$) are known to attacker (using m_i)!

Hence, attacker can divide the messages m_i into two types:

- Type 0 ('no') m_i: y subtraction in line 4 multiply will NOT be performed at first loop iteration ($j = k - 1$).
- Type 1 ('yes') m_i: y subtraction in line 4 multiply will be performed at first loop iteration ($j = k - 1$).

Attack Method ('Differential attack'): Compare average measured total exec. time $\bar{\tau}_0$ for m_i's where subtraction will not be performed, to average total run-time $\bar{\tau}_1$ for remaining m_i's (with subtraction performed).

- If $d_{k-1} = 1$ (line 4 executed at iteration $j = k - 1$), expect $\bar{\tau}_0$ shorter than $\bar{\tau}_1$ by average time of substraction.
- Else, if $d_{k-1} = 0$, (line 4 not executed at iteration $j = k - 1$), expect $\bar{\tau}_0 \approx \bar{\tau}_1$.

Then repeat method for line 4 at iteration $j = k - 2, \ldots, 0$, to obtain rest of bits of d, bit-by-bit!

Timing Side Channels: RSA Signature Generation

A: Timing attack (Summary):

We assume that the attacker already knows $d_{k-1}, \ldots, d_{k-n+1}$. Her goal is to recover the value of the next bit, d_{k-n}.

1. The attacker *guesses* that $d_{k-n} = 1$.
2. Next, the attacker randomly chooses t messages, $m_1, \ldots, m_t$, and prepares two sets of messages, $\mathscr{S}_0$ and $\mathscr{S}_1$, given by

$$\mathscr{S}_0 = \{m_i \mid \text{Montgomery multiplication } R_0 \leftarrow R_0 \cdot R_1 \text{ in Line 4 of Algorithm 5 does not induce a subtraction for } j = k-n\}$$

and

$$\mathscr{S}_1 = \{m_i \mid \text{Montgomery multiplication } R_0 \leftarrow R_0 \cdot R_1 \text{ in Line 4 of Algorithm 5 induces a subtraction for } j = k-n\} \ .$$

3. For each message in set $\mathscr{S}_0$, the attacker requests the signature and measures the computation time to get it. She does the same for messages in set $\mathscr{S}_1$. Let $\bar{\tau}_0$ and $\bar{\tau}_1$ denote the average time (per signature request) for messages in $\mathscr{S}_0$ and $\mathscr{S}_1$, respectively.
4. If $\bar{\tau}_1 \approx \bar{\tau}_0$ then the guess of the attacker was wrong and $d_{k-n} = 0$. If $\bar{\tau}_1 \gg \bar{\tau}_0$ (more precisely, if the time difference between $\bar{\tau}_1$ and $\bar{\tau}_0$ is roughly the time of a subtraction) then the attacker correctly guessed that $d_{k-n} = 1$.
5. Now that the attacker knows $d_{k-1}, \ldots, d_{k-n+1}, d_{k-n}$, she iterates the attack to recover the value of d_{k-n-1} and so on.

Power Side Channels: Simple Power Analysis

In some situations, attacker is able to measure electrical power consumption of attacked device versus time.

- Common example: Attacker controlled Smartcard reader.

Fact: Instantaneous Power consumption of CPUs depends on instruction and data manipulated!

- Basis for power consumption side channel attacks!

Exact dependence depends on chip technology.

Common example (CMOS technology): Significant power is consumed by a bit register only when bit state is flipped from 0 to 1 or 1 to 0.

- **Consequence:** Hamming-Distance (HD) power consumption model: power consumption in computation from state_{i-1} to state_i depends on $HW(\text{state}_{i-1} \oplus \text{state}_i)$ (where HW denotes Hamming Weight).

Another Common Example: Hamming-Weight (HW) power consumption model: power consumption of computation with output data_i depends on $HW(\text{data}_i)$ (where HW denotes Hamming Weight).

- e.g. HD model with data_i loaded into an (initially zero) output CPU register.

Power Side Channels: Simple Power Analysis

Power Analysis: first example – Reverse Engineering Code

Suppose an 8-bit smartcard CPU loads card input byte $x \in \{0, \ldots, 255\}$ and applies some unknown instruction δ to process x.

Attacker goal: recover $\delta \in \{0, \ldots, 255\}$ (reverse engineering).

Attack Idea:

- CPU accumulator state changes from $\text{state}_{i-1} = x$ to $\text{state}_i = \delta$ when processing x with δ.
- Hence (assuming HW model), expect power consumption during processing to depend on $HW(x \oplus \delta)$

Q1: How to determine at what instances of time the CPU is processing input x?

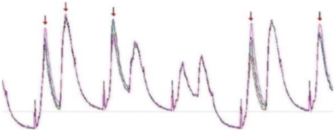
Power Side Channels: Simple Power Analysis

Power Analysis: first example – Reverse Engineering Code

A1: Attack Method to determine at what instances of time the CPU is processing input x:

- Run smartcard on different inputs $x \in \{0, \ldots, 255\}$.
- For each input x, record power consumption vs. time curve.
- Plot power-time graphs for different x's, observe times where graphs differ – hence identify times when x (or function thereof) is processed.

Example measured Power-Time graphs for several inputs x:



Q2: How to use measured power at instants when x is processed by instruction δ to determine δ?

Power Side Channels: Simple Power Analysis

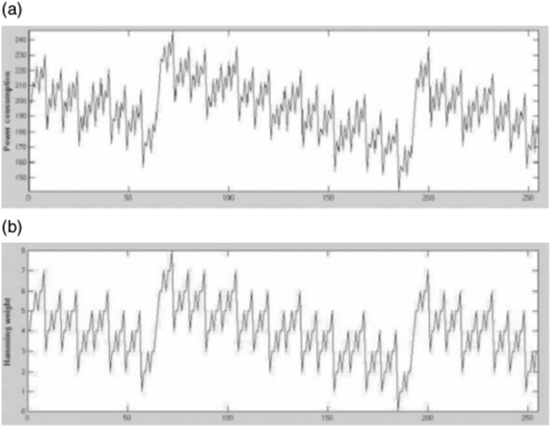
Power Analysis: first example – Reverse Engineering Code - part 2
Recall: (assuming HD model), expect power consumption during processing to depend on $HW(x \oplus \delta)$.
Hence: Graph of $HW(x \oplus \delta)$ versus x should be correlated with Power (at processing instants) versus x:

A2: Attack Method to determine instruction δ from power at instant when x processed:

- Run smartcard on different inputs $x \in \{0, \ldots, 255\}$.
- Plot graph of $P(x)$: power versus x at instant of processing x (as indentified from part 1).
- For each candidate instruction opcode $\delta \in \{0, \ldots, 255\}$, plot $HW_{\delta}(x) = HW(x \oplus \delta)$ versus x.
- Pick as estimate for δ the value for which graphs $HW_{\delta}(x)$ and $P(x)$ are most correlated (similar shape)!

Power Analysis: first example – Reverse Engineering Code - part 2

Example measured $P(x)$ (top) and most correlated $HW_\delta(x) = HW(x \oplus \delta)$ for $\delta = 184$ (bottom):



Power Side Channels: Simple Power Analysis

Power Analysis: second example – RSA Signature Generation
In 'square and multiply' algorithm, presence or absence of multip. step can be used to read off secret key from Power-Time graph!
Example measured Power-Time graph for smartcard running 'square and multiply' RSA signature generation:

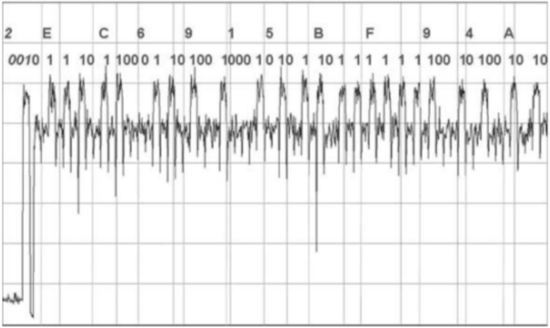


Fig. 13.3 Power trace of an RSA exponentiation.

Power Side Channels: Differential Power Analysis

Q: How to recover a secret key by power analysis if instruction does not directly depend on secret?

A: Identify a place where computation depends on both key portion and input, and use a 'differential power analysis (DPA) attack'!

Example – DPA Attack on AES: Recall AES-128:

- Input 128-bit plaintext block m_i and 128-bit key K are viewed as 4×4 matrices of bytes:

$$m_i = (s^{(i)}_{u,v})_{0 \le u \le 3, 0 \le v \le 3}, K = (k_{u,v})_{0 \le u \le 3, 0 \le v \le 3}.$$

- Processing of plaintext block m_i (repeated in 10 rounds):
  - AddRoundKey: Replaces each byte $s^{(i)}_{u,v}$ with $s^{(i)}_{u,v} \oplus k_{u,v}$.
  - SubBytes: Replaces each byte $s^{(i)}_{u,v}$ with $S_{RD}(s^{(i)}_{u,v})$ where S_{RD} is AES's 8-bit non-linear S-box permutation.
  - ShiftRows : Cyclic shifting of 32-bit state matrix rows.
  - MixColumns : Linear mixing of 32-bit columns of state matrix.

Focus in this attack on first two steps (bytewise)!

Power Analysis: third example – Differential Power Analysis of AES

DPA Attack idea:

- Obtain Power-Time traces $P_i(t)$ for AES encryption on many random input messages $m_i = (s_{u,v}^{(i)})_{0 \leq u \leq 3, 0 \leq v \leq 3}$ for $i = 1, \ldots, t$.
- For each message m_i, byte $\tilde{s}_{u,v}^{(i)}$ of state after first AddRoundKey and SubBytes operations depends on key byte $k_{u,v}$ and input byte $s_{u,v}^{(i)}$:
$$\tilde{s}_{u,v}^{(i)} = S_{RD}(s_{u,v}^{(i)} \oplus k_{u,v})$$
- If attacker guesses $k_{u,v}$ correctly, he knows what the internal state byte $\tilde{s}_{u,v}^{(i)}$ would be!
- Hence attacker knows, e.g. for which m_i's, hamming weight is 'large' (large $P_i(t)$ in HW model at computation instant) or 'small'.

Power Analysis: third example – Differential Power Analysis of AES

Assume HW model. Using its guess for $k_{u,v}$ and based on value of (say) LS bit of $\tilde{s}_{u,v}^{(i)}$, attacker can divide the messages m_i into two types:

- $i \in S_0$ – Type 0 ('low' average HW $\tilde{s}_{u,v}^{(i)}$) m_i: LSbit($\tilde{s}_{u,v}^{(i)}$)=0.
- $i \in S_1$ – Type 1 ('high' average HW $\tilde{s}_{u,v}^{(i)}$) m_i: LSbit($\tilde{s}_{u,v}^{(i)}$)=1.

Attack Method ('Differential attack'): Compare average $\bar{P}_0(t)$ of Power-time graphs $P_i(t)$ over type 0 messages m_i ($i \in S_0$) to average $\bar{P}_1(t)$ of Power-time graphs $P_i(t)$ over type 1 messages m_i ($i \in S_1$):

- If guess of key byte $k_{u,v}$ is correct, expect $\bar{P}_0(t)$ smaller than $\bar{P}_1(t)$ for t=instant of $\tilde{s}_{u,v}^{(i)}$ computation, whereas $\bar{P}_0(t) \approx \bar{P}_1(t)$ for other times t.
- Else, if guess of key byte $k_{u,v}$ is NOT correct, expect $\bar{P}_0(t) \approx \bar{P}_1(t)$ for all times t (why?)

Power Side Channels: Differential Power Analysis

Power Analysis: third example – Differential Power Analysis of AES

Attack summary: For each candidate $k_{u,v}$ for key byte, attacker plots the power difference vs. time graph $\Delta_P(t) \stackrel{\text{def}}{=} \bar{P}_0(t) - \bar{P}_1(t)$ and looks for peaks!

- If no significant peaks in $\Delta_P(t)$, reject candidate $k_{u,v}$ (wrong guess) and move to next candidate.
- When correct $k_{u,v}$ key byte found, repeat to find all other 15 key bytes (each key byte can be found with ≤ 256 trials).

Example Results:

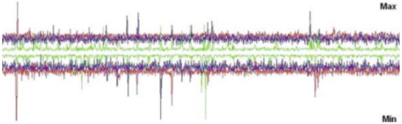


Fig. 13.5 DPA traces for different selection bits.

Cache Side Channels

Idea: Exploit security vulnerabilities due to hardware architecture efficiency features!

Example: Cache memory in modern CPUs (only briefly mention, see Ch. 18 of CryptoEng Book for more).

- Cache is relatively small but fast memory inside CPUs.
- Used to speed up memory access for commonly used values.

Basic ideas:

- When a main memory location is accessed, CPU copies it to fast cache (replacing, e.g. least used old cache value).
- Subsequent accesses of that memory address are fetched quickly from cache copy – cache hit (instead of main memory).
- Memory accesses to addresses not in the cache are fetched slowly from main memory – cache miss.

But this means... a timing side-channel!!

Q: How can it be exploited?

Example A: Cache timing Attacks on AES with lookup table implementation of SubBytes S-box.

Ideas:

- Fast implementations of AES store 8-bit S-box as a lookup table in memory.
- To evaluate SubBytes(x), query memory address x to fetch stored value of SubBytes(x).
- **Vulnerability:** in AES first two rounds, $x = s_{u,v} \oplus k_{u,v}$, where $s_{u,v}$ is input plaintext byte and $k_{u,v}$ is key byte – depends on known input and unknown key byte!
- **Exploit to get info. on key:** Consider two plaintext bytes $s_{u,v}, s_{u',v'}$ and corresponding key bytes $k_{u,v}, k'_{u,v}$. The corresponding memory lookup addresses are:

$$x = s_{u,v} \oplus k_{u,v} \text{ and } x' = s_{u',v'} \oplus k_{u',v'}.$$

- Likely to have a cache hit in SubBytes lookup of x' after SubBytes lookup x for adjacent byte if: $x' = x$, or $s_{u,v} \oplus s_{u',v'} = k_{u,v} \oplus k_{u',v'}$.
- Attack: Guess a candidate value δ_k for $k_{u,v} \oplus k_{u',v'}$. Compare average encryption run-time for many inputs with $s_{u,v} \oplus s_{u',v'} = \delta_k$.
- Correct choice of δ_k will show up as faster average run-time (one more cache hit than for incorrect choices of δ_k on average!).

Compression 'Side Channel'

Compression and Encryption don't naturally Mix!

To reduce communication, common to compress data before sending it. To hide the compressed data, common to encrypt it. **But...**, the length of compressed data reveals information on original data.

- Encryption (by default) does not hide message length.
- Hence: length of encrypted compressed data leaks information on original data!

Q: Can it be exploited in practice?

A [DR12]: In many cases, yes, especially if attacker can mount a chosen plaintext attack!

Compression 'Side Channel': CRIME/BREACH attack on TLS/SSL

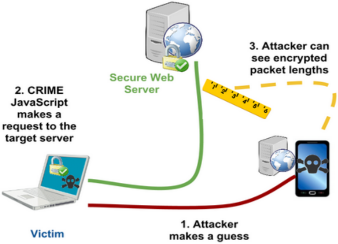
CRIME attack on HTTPS (TLS/SSL) ([Duong-Rizzo 2012]):

- Attacker goal: Steals secret user's cookie with twitter.com
- Attacker installs Javascript on user's browser (user visits attacker's website).
- Attacker guesses first char. of user's cookie, measures length of user's encrypted compressed request
- If guess is correct, compression will reduce length of request ciphertext, else will not!
- Move to guess remaining chars, one by one!

Compression 'Side Channel': CRIME/BREACH attack on TLS/SSL

GET /twid=a
Host: twitter.com
User-Agent: Chrome
Cookie: twid=secret
...
GET /twid=s
Host: twitter.com
User-Agent: Chrome
Cookie: twid=secret

CRIME in a slide



Countermeasure: Disable Compression in SSL/TLS!

Side Channels: Countermeasures

Devising effective countermeasures against side channel attacks is often non-trivial and subject of a large body of research. Will not study in detail (See CryptoEng book for many pointers).

Common approaches:

- Reduce/Eliminate side-channel leakage, e.g.:
  - Use constant time operations
  - Avoid secret-conditioned code execution/branching
- Introducing noise/randomization, e.g.:
  - wait states in hardware
  - data 'masking'/blinding in software, e.g. randomize RSA signature generation as:

$$[(\mu(m) + r_1 \cdot N)^{d + r_2 \cdot \phi(N)} \bmod r_3 N] \bmod N,$$

    with random integers r_1, r_2, r_3 chosen independently for each signature generation.