CSE 207B: Applied Cryptography

Nadia Heninger

UCSD

Spring 2022 Lecture 19

Some slides from Deian Stefan
Announcements

1. HW 8 is due Thursday!
Last time:
  • Lattice-based cryptanalysis

This time:
  • Side-channel attacks
Side-Channel Attacks and Cryptography

Cryptographic program execution can leak information about secrets.

This is mostly outside of the security models we’ve considered, which focus on indistinguishability of ciphertexts and not leakage of keys.

Side channels we’ve already seen: error messages.
Different types of side channels

Computers are physical objects, so measuring them during program execution can reveal information about the program or data.

- Electromagnetic radiation
  - Voltage running through a wire produces a magnetic field
- Power consumption
  - Different paths through a circuit might consume different amounts of power
- Sound (acoustic attacks)
  - Capacitors discharging can make noises
- Timing
  - Different execution time due to program branches
  - Cache timing attacks
- Error messages
  - Error messages might reveal secret information to an attacker
- Fault attacks
  - Induce glitch in circuit to cause incorrect computation
Timing Attacks on Modular Exponentiation
Kocher 96

Pseudocode for “square and multiply” \( c^d \mod N \):

\[
m = 1 \\
\text{for } i = 0 \ldots \text{len}(d): \\
\quad \text{if } d[i] = 1: \\
\quad\quad m = c \times m \mod N \\
\quad m = \text{square}(m) \mod N \\
\text{return } m
\]

• Potential information leaks via multiply, via square, and via modular reduction.
Timing Attacks on Modular Exponentiation
Kocher 96

Attack model: Attacker can query RSA decryption/signature oracle and measure decryption/signing time on chosen messages.

\[
m = 1 \\
\text{for } i = 0 \ldots \text{len}(d): \\
\quad \text{if } d[i] = 1: \\
\quad \quad m = c \times m \mod N \\
\quad m = \text{square}(m) \mod N \\
\text{return } m
\]

To exploit modular reduction:
1. Measure many decryptions/signatures.
2. For each bit \( i \) of secret key, group sample ciphertexts \( c \) into classes: For bit \( i \), is \( cm > N \) (extra modular reduction) or \( cm < N \) (no extra modular reduction)
3. For large enough number of samples, expect to see timing difference depending on whether bit \( i \) was 1 or 0.
Simple power analysis attacks plot power consumption over time. The textbook square and multiply implementation clearly leaks secret key bits in a power trace.

Fig. 11 SPA leaks from an RSA implementation
Basic countermeasures

To avoid timing attacks, need to use a constant-time modular exponentiation algorithm. This is really hard to get right.

*Simple idea: Square and always multiply.*

- Has computational cost from dummy operations. Might still be vulnerable to cache attacks.

\[
c^d \mod N
\]

\[
\text{compute } c^r = r^e c \mod N \text{ for random } r, \text{ evaluate } c^d r \mod N, \text{ and multiply by } r^{-1}.
\]

- Removes correlations between ciphertexts, but doesn't solve secret-dependent code paths.

**RSA blinding.**

\[
c^d \mod N
\]

\[
\text{compute } c^r = r^e c \mod N \text{ for random } r, \text{ evaluate } c^d r \mod N, \text{ and multiply by } r^{-1}.
\]

- Helps remove secret dependence, commonly implemented.

**Exponent blinding.**

\[
c^d \mod N
\]

\[
\text{compute } d^b = d^r + r \phi N \text{ for random } r, \text{ and evaluate } c^d^b \mod N.
\]

- Helps remove secret dependence, commonly implemented.

**Montgomery reduction.**

- Montgomery's modular multiplication algorithm avoids conditional modular reduction step.

- Sequence of operations is fixed.

- Commonly implemented.
Basic countermeasures

To avoid timing attacks, need to use a constant-time modular exponentiation algorithm. This is really hard to get right.

**Simple idea: Square and always multiply.**

- Has computational cost from dummy operations. Might still be vulnerable to cache attacks.

**RSA blinding.**

- To compute \( c^d \mod N \), first compute \( c_r = r^e c \mod N \) for random \( r \), evaluate \( c_r^d \mod N \), and multiply by \( r^{-1} \).
- Removes correlations between ciphertexts, but doesn’t solve secret-dependent code paths.

**RSA exponent blinding.**

- To exponentiate by \( d \), first compute \( d_b = d + r \phi N \) for random \( r \), and evaluate \( c^{d_b} \mod N \).
- Helps remove secret dependence, commonly implemented.

**Montgomery reduction.**

- Montgomery’s modular multiplication algorithm avoids conditional modular reduction step.
- Sequence of operations is fixed.
- Commonly implemented.
Basic countermeasures

To avoid timing attacks, need to use a constant-time modular exponentiation algorithm. This is really hard to get right.

Simple idea: Square and always multiply.

- Has computational cost from dummy operations. Might still be vulnerable to cache attacks.

RSA blinding.

- To compute $c^d \mod N$, first compute $c_r = r^e c \mod N$ for random $r$, evaluate $c_r^d \mod N$, and multiply by $r^{-1}$.
- Removes correlations between ciphertexts, but doesn’t solve secret-dependent code paths.

Exponent blinding.

- To exponentiate by $d$, first compute $d_b = d + r\phi N$ for random $r$, and evaluate $c^{d_b} \mod N$.
- Helps remove secret dependence, commonly implemented.
Basic countermeasures

To avoid timing attacks, need to use a constant-time modular exponentiation algorithm. This is really hard to get right.

Simple idea: Square and always multiply.

- Has computational cost from dummy operations. Might still be vulnerable to cache attacks.

RSA blinding.

- To compute $c^d \mod N$, first compute $c_r = r^e c \mod N$ for random $r$, evaluate $c_r^d \mod N$, and multiply by $r^{-1}$.
- Removes correlations between ciphertexts, but doesn’t solve secret-dependent code paths.

Exponent blinding.

- To exponentiate by $d$, first compute $d_b = d + r\phi N$ for random $r$, and evaluate $c^{d_b} \mod N$.
- Helps remove secret dependence, commonly implemented.

Montgomery reduction.

- Montgomery’s modular multiplication algorithm avoids conditional modular reduction step.
- Sequence of operations is fixed.
In a cache attack, an attack program runs on the same processor as a victim program.

The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory

<table>
<thead>
<tr>
<th>Set</th>
<th>Addr</th>
<th>Cached Data ~64B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F0016280 31C6F4C0 339DD740 614F8480</td>
<td>B5 F5 80 21 E3 2C 9A DA 59 11 48 F2  C7 D7 A0 86 67 18 17 4C 59 B8 58 A7</td>
</tr>
<tr>
<td>1</td>
<td>71685100 132A4880 2A1C0700 C017E9C0</td>
<td>27 BD 5D 2E 84 29 30 B2 8F 27 05 9C 9E C3 DA EE B7 D9 D1 76 16 54 51 5B</td>
</tr>
<tr>
<td>2</td>
<td>311956C0 002D47C0 91507E80 55194040</td>
<td>0A 55 47 82 86 4E C4 15 4D 78 B5 C4 60 D0 2C DD 78 14 DF 66 E9 D0 11 43</td>
</tr>
<tr>
<td>3</td>
<td>9B27F8C0 8E771100 A001FB40 317178C0</td>
<td>84 A0 7F C7 4E BC 3B 0B 20 0C DB 58 29 D9 F5 6A 72 50 35 82 CB 91 78 8B</td>
</tr>
<tr>
<td>4</td>
<td>6618E980 BA0CDB40 89E92C00 090F9C40</td>
<td>35 11 4A E0 2E F1 B0 FC 5A 20 D0 7F 1C 50 A4 F8 6B BB 71 ED 16 07 1F</td>
</tr>
</tbody>
</table>

• In a cache attack, an attack program runs on the same processor as a victim program.
• The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory

• In a cache attack, an attack program runs on the same processor as a victim program.
• The attack program measures memory access times to determine which data the victim loaded into cache.
In a cache attack, an attack program runs on the same processor as a victim program.
The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory

In a cache attack, an attack program runs on the same processor as a victim program.

The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory.

<table>
<thead>
<tr>
<th>Set</th>
<th>Addr</th>
<th>Cached Data ~64B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F0016280 31C6F4C0 339DD740 614F8480</td>
<td>B5 F5 80 21 E3 2C..</td>
</tr>
<tr>
<td>1</td>
<td>71685100 132A4880 2A1C0700 C017E9C0</td>
<td>27 BD 5D 2E 84 29..</td>
</tr>
<tr>
<td>2</td>
<td>311956C0 002D47C0 91507E80 55194040</td>
<td>0A 55 47 82 86 4E..</td>
</tr>
<tr>
<td>3</td>
<td>9B27F8C0 8E771100 A001FB40 317178C0</td>
<td>84 A0 7F C7 4E BC..</td>
</tr>
<tr>
<td>4</td>
<td>6618E980 BA0CDB40 89E22C00 090F9C40</td>
<td>35 11 4A EO 2E F1..</td>
</tr>
</tbody>
</table>

• In a cache attack, an attack program runs on the same processor as a victim program.
• The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory

<table>
<thead>
<tr>
<th>Set</th>
<th>Addr</th>
<th>Cached Data ~64B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F0016280, 31C6F4C0, 339DD740, 614F8480</td>
<td>B5 F5 80 21 E3 2C..</td>
</tr>
<tr>
<td></td>
<td>132A4880, 2A1C0700</td>
<td>27 BD 5D 2E 84 29..</td>
</tr>
<tr>
<td></td>
<td>C017E9C0</td>
<td>2D 1B 6C 5A 39..</td>
</tr>
<tr>
<td>1</td>
<td>71685100, 132A4880, 2A1C0700, C017E9C0</td>
<td>0A 55 47 82 86 4E..</td>
</tr>
<tr>
<td></td>
<td>002D47C0, 91507E80, 55194040</td>
<td>C4 15 4D 78 B5 C4..</td>
</tr>
<tr>
<td></td>
<td>0A 55 47 82 86 4E..</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>311956C0, 91507E80, 55194040</td>
<td>D5 0D 2C DD 78 14..</td>
</tr>
<tr>
<td></td>
<td>002D47C0, 91507E80, 55194040</td>
<td>6F 6E 9D 11 43..</td>
</tr>
<tr>
<td></td>
<td>9B27F8C0, 8E771100, A001FB40, 317178C0</td>
<td>35 11 4A E0 2E F1..</td>
</tr>
<tr>
<td></td>
<td>6618E980, B40CD840, 89E92C00, 090F9C40</td>
<td>80 FC 5A 20 D0 7F..</td>
</tr>
</tbody>
</table>

- In a cache attack, an attack program runs on the same processor as a victim program.
- The attack program measures memory access times to determine which data the victim loaded into cache.
In a cache attack, an attack program runs on the same processor as a victim program.

The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory.

- In a cache attack, an attack program runs on the same processor as a victim program.
- The attack program measures memory access times to determine which data the victim loaded into cache.

### Diagram

**CPU**
- Sends address, Receives data
  - Addr: 2A1C0700
  - Data: 9E C3 DA EE B7 D3...

**Address:** 132E1340

**Data:** 9E C3 DA EE B7 D9...

**Fast**
- hash(addr) to map to cache set

**MEMORY CACHE**

<table>
<thead>
<tr>
<th>Set</th>
<th>Addr</th>
<th>Cached Data ~64B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F0016280, 31C6F4C0, 339DD740, 614F8480</td>
<td>B5 F5 80 21 E3 2C...</td>
</tr>
<tr>
<td>1</td>
<td>71685100, 132A4880, 2A1C0700, C017E9C0</td>
<td>2B 7D 5D 2E 84 29...</td>
</tr>
<tr>
<td>2</td>
<td>311956C0, 002D47C0, 91507E80, 55194040</td>
<td>0A 55 47 82 86 4E...</td>
</tr>
<tr>
<td>3</td>
<td>9B27F8C0, 8E771100, A001FB40, 317178C0</td>
<td>84 A0 7F C7 4E BC...</td>
</tr>
<tr>
<td>4</td>
<td>6618E980, BA0CDB40, 89E92C00, 090F9C40</td>
<td>35 11 4A E0 2E F1...</td>
</tr>
</tbody>
</table>

**Main Memory**
- Big, slow e.g. 16GB SDRAM
  - Address: 132E1340
  - Data: AC 99 17 8F 44 09...
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory.

- In a cache attack, an attack program runs on the same processor as a victim program.
- The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory.

- In a cache attack, an attack program runs on the same processor as a victim program.
- The attack program measures memory access times to determine which data the victim loaded into cache.
Memory caches and cache attacks

Caches hold local (fast) copy of recently-accessed 64-byte chunks of memory

<table>
<thead>
<tr>
<th>Set</th>
<th>Addr</th>
<th>Cached Data ~64B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F0016280 31C68400 339D7D40 614F8480</td>
<td>B5 F5 80 21 E3 2C..</td>
</tr>
<tr>
<td>1</td>
<td>71685100 132A4880 2A1C0700 C017E9C0</td>
<td>27 BD 5D 2E 84 29..</td>
</tr>
<tr>
<td>2</td>
<td>311956C0 002D47C0 91507E80 5519D040</td>
<td>0A 55 47 82 86 4E..</td>
</tr>
<tr>
<td>3</td>
<td>9B27F8C0 8E771100 A001FB40 132E1340</td>
<td>84 A0 7F C7 4E BC..</td>
</tr>
<tr>
<td>4</td>
<td>6618E980 BA0CDB40 89E92C00 09099C40</td>
<td>35 11 4A E0 2E F1..</td>
</tr>
</tbody>
</table>

In a cache attack, an attack program runs on the same processor as a victim program.

The attack program measures memory access times to determine which data the victim loaded into cache.
In a cache attack, an attack program runs on the same processor as a victim program. The attack program measures memory access times to determine which data the victim loaded into cache.
In a cache attack, an attack program runs on the same processor as a victim program.

The attack program measures memory access times to determine which data the victim loaded into cache.
Cache attacks against cryptography

Attack model:

• Attacker can run code on same physical hardware as victim program.
• Attacker can profile victim code to get layout of program memory.
• Attacker can time cryptographic operations from victim.
• Attacker evicts chosen data from cache and triggers victim execution: if it’s slower, victim must have tried to access evicted data.
Cache Attacks against AES

Bernstein 2005

- Recall AES algorithm consists of xors, shifts, S-box.
- For speed, operations can be precomputed as a lookup table “T-table”
- Table queries dependent on key values
- A cache attack can reveal the lookup locations and thus the secret key
Cache attacks against modular exponentiation

Cache attack on instructions
If square and multiply functions in different cache lines, interrupt execution at a precise time and see if multiply function was accessed at bit $i$. 
Cache attacks against modular exponentiation

**Fixed-window exponentiation**

Write \( d \) in base \( 2^w \): \( d = (d_0, d_1, \ldots, d_n) \)

precompute \( c^0, c^1, \ldots, c^{w-1} \)

\( m = 1 \)

for \( i = 0 \ldots n \):

  for \( j = 1 \ldots w \):

    \( m = \text{square}(m) \mod N \)

    \( m = \text{table}[d_i] \ast m \mod N \)

return \( m \)

Cache attack against windowed exponentiation: Read off digits of \( d \) from accessed lookup table.
Fault Attacks

Many ways to glitch a computer or a circuit during computation:

• Shine a powerful laser on the circuit
• Fluctuate power
• Rowhammer
• Heat RAM up
• Wait for cosmic rays
Fault attacks on RSA-CRT
[ Boneh DeMillo Lipton 1997 ]

Recall: RSA implementations use the Chinese Remainder Theorem:
1. Precompute \( d_p = d \mod (p - 1) \) and \( d_q = d \mod q - 1 \).
2. To generate a signature on \( m \), compute

\[
s_p = m^{d_p} \mod p \quad s_q = m^{d_q} \mod q
\]

3. Use the Chinese remainder theorem to construct \( s \mod N \).
Fault attacks on RSA-CRT

[Boneh DeMillo Lipton 1997]

Recall: RSA implementations use the Chinese Remainder Theorem:
1. Precompute $d_p = d \mod (p - 1)$ and $d_q = d \mod q - 1$.
2. To generate a signature on $m$, compute
   
   $$s_p = m^{d_p} \mod p \quad s_q = m^{d_q} \mod q$$

3. Use the Chinese remainder theorem to construct $s \mod N$.

If a fault is induced during the computation of $s_p$, then

$$m \not\equiv s^e \mod p \quad m \equiv s^e \mod q$$

So

$$\gcd(m - s^e, N) = q$$
Countermeasures against fault attacks

Validate results of computations after they are computed.
Memory-stealing attacks

• Speculative execution attacks (Spectre, Meltdown, variants) are side-channel attacks; allow unprivileged processes to read memory contents using a cache attack.
• “Cold boot” attacks allow a physical attacker to read contents of RAM by exploiting DRAM remanence.

In some sense these attacks are too powerful: allow near error-free access to arbitrary memory contents. Countermeasures needed at hardware and not cryptography level.
Summary: side-channel attacks

• Multi-decade arms race in developing and defending against side-channel attacks.

• Hardware development continues to open new side channel opportunities: Intel SGX, speculative execution, new cache structures, RAM structure (Rowhammer)

• Side-channel attacks tend to get more attention than other security threats