Side-channels

Tom Ristenpart
CS 6431
Recall from last time:

Pick target(s)

Choose launch parameters for malicious VMs

Each VM checks for co-residence

Frequently achieve advantageous placement

This shouldn’t matter if VMM provides good isolation!
Today

- Flush+Reload side-channel attack
- RSA remote side-channel attack
RSA reminder

\[ Z_N^* = \{ i \mid \gcd(i,N) = 1 \} \quad \text{gcd} \quad \phi(N) = |Z_N| \]

Claim: Suppose \( e,d \in Z_{\phi(N)}^* \) satisfying \( ed \mod \phi(N) = 1 \) then for any \( x \in Z_N^* \) we have that
\[
(x^e)^d \mod N = x
\]

\[
(x^e)^d \mod N = x^{(ed \mod \phi(N))} \mod N = x^1 \mod N = x \mod N
\]

First equality is by Euler’s Theorem
RSA reminder

\[ \mathbb{Z}_N^* = \{ i \mid \gcd(i, N) = 1 \} \quad \text{and} \quad \phi(N) = |\mathbb{Z}_N^*| \]

Claim: Suppose \( e, d \in \mathbb{Z}_{\phi(N)}^* \) satisfying \( ed \mod \phi(N) = 1 \) then for any \( x \in \mathbb{Z}_N^* \) we have that

\[ (x^e)^d \mod N = x \]

\[ \mathbb{Z}_{15}^* = \{ 1, 2, 4, 7, 8, 11, 13, 14 \} \quad \mathbb{Z}_{\phi(15)}^* = \{ 1, 3, 5, 7 \} \]

\( e = 3, \ d = 3 \) gives \( ed \mod 8 = 1 \)

<table>
<thead>
<tr>
<th>( x )</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>7</th>
<th>8</th>
<th>11</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x^3 \mod 15 )</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>13</td>
<td>2</td>
<td>11</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>( (x^3)^3 \mod 15 )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>11</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>
The RSA trapdoor permutation

pk = (N,e)  \quad sk = (N,d) \quad \text{with } ed \mod \phi(N) = 1

f_{N,e}(x) = x^e \mod N \quad g_{N,d}(y) = y^d \mod N
The RSA trapdoor permutation

\[ pk = (N, e) \quad sk = (N, d) \quad \text{with} \quad ed \mod \phi(N) = 1 \]

\[ f_{N,e}(x) = x^e \mod N \quad g_{N,d}(y) = y^d \mod N \]

But how do we find suitable \( N, e, d \)?

If \( p, q \) distinct primes and \( N = pq \) then \( \phi(N) = (p-1)(q-1) \)

Why?

\[
\phi(N) = |\{1, \ldots, N-1\}| - |\{ip : 1 \leq i \leq q-1\}| - |\{iq : 1 \leq i \leq p-1\}|
\]
\[
= N-1 - (q-1) - (p-1)
\]
\[
= pq - p - q + 1
\]
\[
= (p-1)(q-1)
\]
The RSA trapdoor permutation

\[ \text{pk} = (N,e) \quad \text{sk} = (N,d) \quad \text{with} \quad ed \mod \phi(N) = 1 \]

\[ f_{N,e}(x) = x^e \mod N \quad g_{N,d}(y) = y^d \mod N \]

But how do we find suitable \( N,e,d \)?

If \( p,q \) distinct primes and \( N = pq \) then \( \phi(N) = (p-1)(q-1) \)

Given \( \phi(N) \), choose \( e \in \mathbb{Z}_{\phi(N)}^* \) and calculate

\[ d = e^{-1} \mod \phi(N) \]
Inverting RSA: given $N, e, y$ find $x$ such that $x^e \equiv y \pmod{N}$

- EASY because $f^{-1}(y) = y^d \pmod{N}$

  - Know $d$
  - EASY because $d = e^{-1} \pmod{\varphi(N)}$

    - Know $\varphi(N)$
    - EASY because $\varphi(N) = (p-1)(q-1)$

    - Know $p, q$
    - Learning $p, q$ from $N$ is the factoring problem

- We don’t know if inverse is true, whether inverting RSA implies ability to factor
Textbook exponentiation

How do we compute $h^x$ for any $h \in \mathbb{Z}_N^*$?

**Exp(h,x,N)**
- $X' = h$
- For $i = 2$ to $x$ do
  - $X' = X' * h$
- Return $X'$

Requires time $O(|G|)$ in worst case.

**SqrAndMulExp(h,x,N)**
- $b_k,\ldots,b_0 = x$
- $f = 1$
- For $i = k$ down to $0$ do
  - $f = f^2 \mod N$
  - If $b_i = 1$ then
    - $f = f * h \mod N$
- Return $f$

Requires time $O(k)$ multiplies and squares in worst case.
SqrAndMulExp(h, x, N)

\[ b_k, \ldots, b_0 = x \]

\[ f = 1 \]

For \( i = k \) down to 0 do

\[ f = f^2 \mod N \]

If \( b_i = 1 \) then

\[ f = f \cdot h \mod N \]

Return \( f \)

\[ x = \sum_{b_i \neq 0} 2^i \]

\[ h^x = h^{\sum_{b_i \neq 0} 2^i} = \prod_{b_i \neq 0} h^{2^i} \]

\[ h^{11} = h^{8+2+1} = h^8 \cdot h^2 \cdot h \]

\[ b_3 = 1 \quad f_3 = 1 \cdot h \]

\[ b_2 = 0 \quad f_2 = h^2 \]

\[ b_1 = 1 \quad f_1 = (h^2)^2 \cdot h \]

\[ b_1 = 1 \quad f_0 = (h^4 \cdot h)^2 \cdot h = h^8 \cdot h^2 \cdot h \]
SqrAndMulExp(h, x, N)

\[ b_k, \ldots, b_0 = x \]

\[ f = 1 \]

For \( i = k \) down to 0 do

\[ f = f^2 \mod N \]

If \( b_i = 1 \) then

\[ f = f \cdot h \mod N \]

Return \( f \)

\[ x = \sum_{b_i \neq 0} 2^i \]

\[ h^x = h^{\sum_{b_i \neq 0} 2^i} = \prod_{b_i \neq 0} h^{2^i} \]

What side-channels might arise?

- Timing
- CPU state (caches, branch predictors,...)
- Power
Attack setting

Co-located placement on cloud instance

RSA-Decrypt takes adversarially supplied ciphertext $c \in \mathbb{Z}_N^*$ and computes $c^d \mod N$

Attacker running on same server, in different VM (or process)

Where would this come up in practice?
(Part of) TLS handshake for RSA transport

Client

Pick random Nc

Check CERT using CA public verification key

Pick random PMS

C <- E(pk, PMS)

Server

ClientHello, MaxVer, Nc, Ciphers/CompMethods

ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

CERT = (pk of bank, signature over it)

C

PMS <- D(sk, C)
Cache-based side channel attacks

- A long literature on cache side channel attacks
  - Percivel 2005: RSA side channels
  - Tromer et al. 2005: AES side channels
- Today: particularly simple one by Yarom and Faulkner useful in PaaS clouds

```
SqrAndMulExp(h, x, N)

b_k, ..., b_0 = x
f = 1

For i = k down to 0 do
  f = f^2 mod N
  If b_i = 1 then
    f = f*h mod N

Return f
```
Towards Prime+Probe

Suppose victim and attacker shares a core

Also sharing L1 instruction & data caches
Prime+Probe protocol

Run (S) operation
- Attacker VM
- Victim VM
- Interrupt
- Scheduling order on CPU core

Run (M) operation
- Attacker VM
- Victim VM
- Interrupt

L1 instr cache (each row represents cache set)

- Fast
- Slow

• Timings correlated to (distinct) cache usage patterns of S, M operations
• Can spy frequently (every ~16 μs) by exploiting scheduling
Prime+Probe limitations

• Originally worked only for L1 caches
  – Some recent work extending to LLC in certain settings
  – Multi-core settings difficult but feasible in lab (Zhang et al. 2012)

• Lots of noise from various sources

• State-of-the-art:
  – Sinan Inci et al. 2016 “Cache Attacks Enable Bulk Key Recovery on the Cloud”
Towards Flush+Reload

Deduplication-based memory page sharing (Linux, KVM, VMWare)

- Duplicate memory pages detected, physical pages coalesced
- Virtual address spaces different, but mapped to same physical addresses

Inclusive cache architecture

Main memory

Libgcrypt square() instructions
Towards Flush+Reload

Deduplication-based memory page sharing (Linux, KVM, VMWare)
- Duplicate memory pages detected, physical pages coalesced
- Virtual address spaces different, but mapped to same physical addresses
Flush+Reload protocol

- Flush from LLC memory line of interest
- Wait
- Time reloading memory line

![Graph showing probe time and access by victim](image)
Attacking Square and Multiply

\[
\text{SqrAndMulExp}(h, x, N)
\]
\[
b_k, \ldots, b_0 = x
\]
\[
f = 1
\]
For \( i = k \) down to 0 do
\[
f = f^2 \mod N
\]
If \( b_i = 1 \) then
\[
f = f \times h \mod N
\]
Return \( f \)
Flush+Reload type attacks damaging

- Immediately used in a large number of follow-up papers to break various things
- Requires memory deduplication or shared libraries
  - Deduplication turned off in Amazon EC2, but available in modern hypervisors
  - Different VMs do not share libraries
- In Linux, shared libraries and deduplication the norm
  - PaaS services vulnerable [Zhang et al. 2014]
General victim execution tracing

```c
#include "stdio.h"
int b;
int inc(int number) {
    return number + 1;
}
int main() {
    int a = 9;
    if (a % 2 == 1)
        a = inc(a);
    b = a;
    return 0;
}
```
```c
#include "stdio.h"
int b;
int main() {
    int a = 9;
    if (a % 2 == 1)
        a = inc(a);
    b = a;
    return 0;
}

int inc(int number) {
    return number + 1;
}
```
Control-Flow Graph

chunk 1: [400480-4004bf]

- 4004b6: mov $0x9,%edi
- 4004bb: callq 4004b4 <inc>

chunk 2: [400300-40033f]

- 400324: lea 0x1(%rdi),%eax
- 400327: retq

chunk 3: [4004c0-4004ff]

- 4004c0: mov %eax,0x200b60(%rip)
- 4004c6: mov $0x0,%eax
- 4004cb: retq
Control-Flow Graph

chunk1
[400480-4004bf]

chunk2
[400300-40033f]

chunk 3
[4004c0-4004ff]
An Attack NFA

chunk1
[400480-4004bf]

Flush-Reload

chunk2
[400300-40033f]

chunk 3
[4004c0-4004ff]

Flush-Reload

{c1}

(c1, t₁, t₂)

{c2, c3}

(c2, t₃, t₄)

(c2, t₃, t₄)

{c3}

(c3, t₅, t₆)

{} (c3, t₇, t₈)

start
Three Example Attacks

• Inferring Sensitive User Data

• SAML-based Single Sign-on Attacks

• Password-Reset Attack
  – Demonstrated successful attacks against Magento (controlled by ourselves) in a public PaaS cloud.
  – After $2^{20}$ offline computation, the attacker can narrow down the password reset token to $2^2$ possible values---easy to brute-force online.
Using timing

Co-located placement on cloud instance

RSA-Decrypt takes adversarially supplied ciphertext $c \in \mathbb{Z}_N^*$ and computes $c^d \mod N$

Attacker can time operation
Prior work before BB: Kocher 96 timing attack

\[ x = b_k b_{k-1} ... b_{k-i+1} u_{k-i} u_{k-i-1} ... u_0 \]

Guess some bits at front Remaining exponent bits are unknown

Predict how long it should take to decrypt (random) value C

- If guess is right, predictions will correlate with observed timings
- If guess is wrong, no correlations
- As more \( b_k \ldots b_{k-i+1} \) known, predictions get stronger

Kocher shows can do with \( q = O(k) \)

\[
\text{SqrAndMulExp}(h,x,N) \\
b_k,\ldots,b_0 = x \\
f = 1 \\
\text{For } i = k \text{ down to } 0 \text{ do} \\
f = f^2 \mod N \\
\text{If } b_i = 1 \text{ then} \\
f = f \cdot h \mod N \\
\text{Return } f
\]
Kocher attack in practice?

• BB point out:
  – OpenSSL widely used implementation of TLS

• Kocher attack doesn’t work against it:
  – OpenSSL uses CRT, sliding windows, two different modular multiplication algorithms
    – CRT: Chinese Remainder Theorem
    – Sliding window: exponentiation by handling exponent in chunks, not one bit at a time
Chinese Remainder Theorem

For \( n = n_1 n_2 ... n_k \) with \( \gcd(n_i, n_j) = 1 \)

System of congruences
\[
x = x_1 \mod n_1 = \ldots = x_k \mod n_k
\]
has exactly one solution and we can find it efficiently

For RSA with \( N = pq \)
\[
d1 = d \mod (p-1)
d2 = d \mod (q-1)
q^{-1} = q^{-1} \mod p
\]
Compute \( m_1 = h^{d1} \mod p \quad m_2 = h^{d2} \mod q \)
Compute \( m = m_1 + (q^{-1} \times (m1 - m2) \mod p) \times q \)
Towards a timing attack

Decryption requires $m_2 = h^{d_2} \mod q$

- Square-and-multiply (lots of multiplies mod $q$)
- Sliding window (fewer multiplies mod $q$)

Multiply $x \cdot y \mod q$ uses Montgomery reduction

- Let $R = 2^z$ for some $z$
- Compute $(xR \cdot yR) \cdot R^{-1} = zR \mod q$
  - Fast to compute $R^{-1}$
  - At end of computation: $zR > q$ then subtract $q$

Schindler’s observation:

$$\Pr[\text{subtract } q] \approx \frac{h \mod q}{2R}$$
The graph shows the number of extra reductions in Montgomery's algorithm as a function of the values of $g$ between 0 and $6q$. There are discontinuities in the graph when:

- $g \mod q = 0$,
- $g \mod p = 0$,
Guess first few bits of \( q = b_k \ldots b_1 \)
Let \( h_{lo} = b_k \ b_{k-1} \ldots b_{k-i+1} \ 0 \ 0 \ 0 \ldots 0 \)
Let \( h_{hi} = b_k \ b_{k-1} \ldots b_{k-i+1} \ 1 \ 0 \ 0 \ldots 0 \)

If \( b_{k-i} = 0 \) then: \( h_{lo} < q < h_{hi} \)
If \( b_{k-i} = 1 \) then: \( h_{lo} < h_{hi} < q \)
Many more details

• **Secondary timing effect**
  – Karatsuba versus standard multiplication
  – Opposite timing difference for $h_{hi} - h_{lo}$
  – Use absolute values: one effect always dominates

• **Sliding window (few multiplications)**
  – Amplify by querying a bunch of values:
    \[ h_{hi}, h_{hi} + 1, h_{hi} + 2, \ldots \]
  – Called a neighborhood
Timing differences measurable

The graph shows the time difference in CPU cycles as a function of neighborhood size. Two curves are plotted:

- A solid line representing the zero-one gap when a bit of $q=0$.
- A dotted line representing the zero-one gap when a bit of $q=1$.

A zero-one gap is indicated by an arrow, which shows a significant difference at certain neighborhood sizes.
The attack worked...

- ~1.5 million queries for 1024 bit RSA
- 2 hours on that era’s computers
- Blinding countermeasure:
  \[
  y = r^e \cdot c \mod N \\
  m = y^d / r \mod N
  \]

```
SqrAndMulExp(h, x, N)

b_k, ..., b_0 = x
f = 1

For i = k down to 0 do
  f = f^2 \mod N
  If b_i = 1 then
    f = f * h \mod N

Return f
```
Side-channel countermeasures

• Constant time code
  – Input-independent memory accesses, timing

• Very difficult to get this!
  – Input-dependent instruction timing on CPU

• Don’t share physical resources
Three Example Attacks

• Inferring Sensitive User Data
• SAML-based Single Sign-on Attacks
• Password-Reset Attack
Password Reset
Password Reset Attack

Library calls

Shared Library

gettimeofday()

Pseudo-
Random Number
Generator

Password Reset

Token Verification

User

Token

Application

Pseudo-
Random Number
Generator

Prediction

Attacker

Token

Application

Shared OS

Attack NFA
Call Graph of Password Resetting
Attacker’s Strategy

Attacker application → Victim application → Attacker’s email account

- Password reset against his own account (attack NFA)
- Offline: Brute force value of getpid()
- Password reset against victim account (attack NFA)
- Online: token guessing
- Password reset token
- (HTTP keepAlive)
The Attack NFA

c1. gettimeofday@plt
T

\(c_2\), \(0\), \(T\)
\(c_1\), \(0\), \(1\)
\(c_2\), \(5\), \(20\)
\(c_1\)

\(c_4\), \(1\), \(30\)
\(c_2\), \(1\), \(T\)
\(c_3\), \(1\), \(T\)
\(c_1\), \(0\), \(1\)

\(c_1\), \(c_4\)
\(c_4\), \(5\), \(30\)
\(c_1\), \(0\), \(1\)
\(c_3\), \(1\), \(T\)
\(c_5\), \(1\), \(T\)
\{c_1, c_4\}
\{c_1, c_4\}
\{c_1, c_4\}

\(c_1\), \(0\), \(1\)
\{\}
\{c_1, c_4\}
\{c_1, c_4\}
\{c_1, c_4\}
\{c_5\}

\(c_4\), \(31\), \(T\)

php5-fpm [0x42ee40 - 0x42ee7f]
php5-fpm [0x5eab00 - 0x5eab3f]
php5-fpm [0x5f0380 - 0x5f03bf]
php5-fpm [0x6028c0 - 0x6028ff]
php5-fpm [0x5eab40 - 0x5eab7f]
Evaluation

• Demonstrated successful attacks against Magento (controlled by ourselves) in a public PaaS cloud.

• After $2^{20}$ offline computation, the attacker can narrow down the password reset token to $2^2$ possible values---easy to brute-force online.