Side Channel Analysis

Chester Rebeiro
IIT Madras
Modern ciphers designed with very strong assumptions

• Kerckhoff’s Principle
  – The system is completely known to the attacker. This includes encryption & decryption algorithms, plaintext
  – only the key is secret

• Why do we make this assumption?
  – Algorithms can be leaked (secrets never remain secret)
  – or reverse engineered

Mallory’s task is therefore very difficult....
Security as strong as its weakest link

- Mallory just needs to find the weakest link in the system....there is still hope!!!
Side Channels
Side Channel Analysis (the weak links)

Side Channels
Eg. Power consumption / radiation of device, execution time, etc.

Mallory
Gets information about the keys by monitoring Side channels of the device
Side Channel Analysis

Mallory measures some physical parameter of the device like radiation, power consumption or timing.

Alice encrypts the message “Attack at Dawn!!”

Radiation from Device

Secret information

| 0 | 0 | 1 | 1 | 1 |

---

CR
# Types of Side Channel Attacks

<table>
<thead>
<tr>
<th></th>
<th>Passive Attacks</th>
<th>Active Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-Invasive Attacks</strong></td>
<td>The device is operated largely or even entirely within its specification.</td>
<td>The device, its inputs, and/or its environment are manipulated in order to make the device behave abnormally.</td>
</tr>
<tr>
<td>Device attacked as is, only accessible interfaces exploited, relatively inexpensive</td>
<td><strong>Side-channel attacks:</strong> timing attacks, power + EM attacks, cache trace</td>
<td>Insert fault in device without depackaging: clock glitches, power glitches, or by changing the temperature</td>
</tr>
<tr>
<td><strong>Semi-Invasive Attacks</strong></td>
<td>Read out memory of device without probing or using the normal read-out circuits</td>
<td>Induce faults in depackaged devices with e.g. X-rays, electromagnetic fields, or light</td>
</tr>
<tr>
<td>Device is depackaged but no direct electrical contact is made to the chip surface, more expensive</td>
<td><strong>Invasive Attacks</strong></td>
<td>Depackaged devices are manipulated by probing, laser beams, focused ion beams</td>
</tr>
<tr>
<td>No limits what is done with the device</td>
<td>Probing depackaged devices but only observe data signals</td>
<td></td>
</tr>
</tbody>
</table>

Source: Elisabeth Oswald, Univ. of Bristol
Timing Attacks
Execution Time

What can you tell from the execution time of this function?

```c
unsigned int Divide(unsigned int a, unsigned int b){
    if (b == 0)
        return ERROR;
    else
        return a/b;
}
```

- Execution time depends on values of a and b
  - Fastest when b=0
  - Varies depending a / b
- Thus information can be inferred from execution time.
  - Can we get secret information from the timing?
Measuring Time Accurately

- **RDTSC**: Read Time Stamp Counter
  - 128 bit register that is reset at boot up and increments at every clock cycle

**Usage**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flush Pipeline</strong></td>
<td></td>
</tr>
<tr>
<td>T1 = rdtsc()</td>
<td></td>
</tr>
<tr>
<td><strong>Flush Pipeline</strong></td>
<td><strong>/// invoke function to be timed</strong></td>
</tr>
<tr>
<td>T2 = rdtsc()</td>
<td></td>
</tr>
<tr>
<td><strong>Flush pipeline</strong></td>
<td></td>
</tr>
</tbody>
</table>
Flush Pipeline and Read TSC

timestamp()

1. cpuid ; ensure preceding instructions complete
2. rdtsc ; read time stamp
3. cpuid ; ensure preceding instructions complete
4. mov time, eax ; move counter into variable
5. load ebx, (epp) ; a load from memory
6. cpuid ; ensure preceding instructions complete
7. rdtsc ; read time stamp again
8. cpuid ; ensure preceding instructions complete
9. sub eax, time ; find the difference

http://arbidprobramming.blogspot.in/2010/05/measuring-timing-accurately-on-intel.html
DIV: Measuring Execution Time

- For randomly chosen values of a/b
- Note the distribution

```c
unsigned int Divide(unsigned int a, unsigned int b) {
    if (b == 0)
        return ERROR;
    else
        return a/b;
}
```
Timing Attacks on RSA
(breaking real-world implementations)

Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other systems

Remote Timing Attacks are Practical
Exponentiation with Square and Multiply

\[ y = x^c \mod n \]

- say, \( x = 45 = (101101)_2 \)

<table>
<thead>
<tr>
<th>i</th>
<th>c</th>
<th>exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>( y )</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>( y^2 )</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>( y^{4+1} = y^5 )</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>( y^{10+1} = y^{11} )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( y^{22} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( y^{44+1} = y^{45} )</td>
</tr>
</tbody>
</table>

**Algorithm** : \( \text{SQUARE-AND-MULTIPLY}(x, c, n) \)

\[
\begin{align*}
  z &\leftarrow 1 \\
  \text{for } i &\leftarrow \ell - 1 \text{ downto } 0 \\
  \quad \text{do } &\begin{cases} 
    z &\leftarrow z^2 \mod n \\
    \text{if } c_i = 1 &\text{then } z &\leftarrow (z \times x) \mod n
  \end{cases}
\end{align*}
\]

\text{return } (z)
The Attack setup

\[ y = x^c \mod n \]

Algorithm: \( \text{SQUARE-AND-MULTIPLY}(x, c, n) \)

\[
\begin{align*}
z & \leftarrow 1 \\
\text{for } i \leftarrow \ell - 1 \text{ downto } 0 \\
& \quad \text{do } \begin{cases} 
z & \leftarrow z^2 \mod n \\
& \text{if } c_i = 1 \\
& \quad \text{then } z \leftarrow (z \times x) \mod n 
\end{cases} \\
\text{return } (z)
\end{align*}
\]
Kocher’s Attack to find the $b^{th}$ bit

**Assumption**: Attacker knows bits $c_{l-1}, c_{l-2} \cdots c_{b+1}$

**Aim**: To discover bit $c_b$

S1. Choose a random $x$

S2. Trigger an encryption to get $y \equiv x^c \mod n$ and execution time $t$

S3. Form $c^{(0)} = (c_{l-1}, c_{l-2} \cdots, c_{b+1}, 0,0)$  

   Trigger an encryption to get $y \equiv x^{c^{(0)}} \mod n$ and execution time $t^{(0)}$

S4. Form $c^{(1)} = (c_{l-1}, c_{l-2} \cdots, c_{b+1}, 1,0)$  

   Trigger an encryption to get $y \equiv x^{c^{(1)}} \mod n$ and execution time $t^{(1)}$

S5. Compute difference in execution time

   $$d^{(0)} = t - t_0$$  
   $$d^{(1)} = t - t_1$$

S6. Repeat from S1 several times

S7. Compute distributions of $D^{(0)}$ from all $d^{(0)}$ and $D^{(1)}$ from all $d^{(1)}$

S8. If $\text{var}(D^{(0)}) < \text{var}(D^{(1)})$ return '$c_b = 0'$

   Else return '$c_b = 1$'
Adding Distributions

• Consider two random variables $G_1$ and $G_2$ with mean and variance $(m_1, v_1)$ and $(m_2, v_2)$
• $G_1 + G_2$ is a distribution with mean and variance $(m_1 + m_2, v_1 + v_2)$
• $G_1 - G_2$ is a distribution with mean and variance $(m_1 - m_2, v_1 + v_2)$
Assumption

- During the square and multiply execution,
- The time taken to perform a square or a multiply is independent of all other square and multiply operations

```
Algorithm : SQUARE-AND-MULTIPLY(x, c, n)

z ← 1
for i ← ℓ - 1 downto 0
    do { if c_i = 1
          then z ← (z × x) mod n
        z ← z^2 mod n }
return (z)
```
Execution Time of Square and Multiply

- Is a Normal Distribution: $T$ with $(m, v)$
- Each iteration by itself is a distribution

$c = (101101)_2$

$T = 3T_{\text{MUL}} + 5T_{\text{SQ}}$

$v = 3v_{\text{MUL}} + 5v_{\text{SQ}}$
4 cases

• Bit $c_b$ in secret is 1
  – Attacker guessed 1 (correctly)
  – Attacker guessed 0 (wrong)

• Bit $c_b$ in secret is 0
  – Attacker guessed 0 (correctly)
  – Attacker guessed 1 (wrong)

what we will see is that when the attacker guess is wrong, then the variance is higher
Case 1.1, when bit $c_b$ is 1

and attacker guess is correct

\[ \nu - \nu^* = \nu_{MUL} + \nu_{SQ} \]

Variance Reduces
Case 1.2, When $c_b$ bit is 1

And attacker guess is wrong

\[ v - v^* = 2v_{MUL} + 3v_{SQ} > (v_{MUL} + v_{SQ}) \]

Variance Increases
Case 2.1, when $c_b$ is 0

And attacker guess is correct

\[ \nu - \nu^* = 1\nu_{MUL} + 1\nu_{SQ} \]

Variance Less
Case 2.2, When $c_b$ is 0

\[ v - v^* = 2v_{MUL} + 3v_{SQ} \]

When guess is wrong

Variance increases
The Iterative Attack

• We start with the MSB and target one bit at a time till we reach the LSB

What happens if there is an error in a bit?
Naïve Countermeasures don’t always work

All operations constant time

Easier said than done!
Practically infeasible
Highly dependent on system architecture
Naïve Countermeasures don’t always work

Adding noise to timing measurements
  – Such as, by random delays

These reduce the Signal-to-noise ratio.

Can be circumvented by taking making more number of measurements
If the SNR reduces by a factor of \( n \), then number of measurements increase by a factor of \( n^2 \)
Prevention by Blinding

choose \( r \) randomly and keep it secret

compute \( r^c \mod n \) and \( r^{-c} \equiv r^c \mod n \)

\[
y' \equiv (x \cdot r)^c \mod n
\]

\[
y \equiv y' \cdot r^{-c} \mod n
\]

The blind ‘\( r \)’ should be changed before each decryption.
One way is to choose \( r \) and compute \( r^2 \).
For the next encryption compute \( r^2 \) and \( (r^{-1})^2 \)

Why does it work?
Since ‘\( r \)’ is secret, attackers have no useful knowledge about the input to the modular exponentiation.
RSA Decryption in Practice
(OpenSSL crypto-lib uses CRT)

1. \[ x_1 \equiv y^{a_1} \mod p \]
2. \[ x_2 \equiv y^{a_2} \mod q \]

where
\[ a_1 \equiv a \mod \phi(p) \]
\[ a_2 \equiv a \mod \phi(q) \]

3. Derive \( x \) from \( x_1 \) and \( x_2 \)
   a. Compute \( q' \equiv q^{-1} \mod p \)
   b. \[ h = q'(x_2 - x_1) \mod p \]
   c. \[ x = x_1 + h \cdot q \]

\( x \) is the message
\( y \) is the ciphertext
\( a \) is the secret key
\( n = pq \)

Garner’s formula.
\[ x = (x_1 \cdot p \cdot p^{-1} \mod q + x_2 \cdot q \cdot q^{-1} \mod p) \mod n \]
from EEA, \[ p \cdot p^{-1} \mod q + q \cdot q^{-1} \mod p = 1 \]
\[ p \cdot p^{-1} \mod q = 1 - q \cdot q^{-1} \mod p \]
\[ x = x_1 + (x_2 - x_1)q \cdot q^{-1} \mod p \]

Crypto libraries like the OpenSSL implement multiplication using
the Montgomery multiplication.
Preventing Kocher’s Attack with the Montgomery Ladder

- \( s = y^c \mod n \)

say, \( c=45=(101101)_2 \)

\[
\begin{array}{|c|c|c|c|}
\hline
 i & c_i & R0 & R1 \\
\hline
 0 & 1 & y & y \\
 1 & 0 & \gamma^2 & \gamma^3 \\
 2 & 1 & \gamma^5 & \gamma^6 \\
 3 & 1 & \gamma^{11} & \gamma^{12} \\
 4 & 0 & \gamma^{22} & \gamma^{23} \\
 5 & 1 & \gamma^{45} & \gamma^{46} \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{Input: } & c, \ y \\
\text{Output: } & y^c \mod n \\
\text{\textbf{exp}(x, y) } & \{} \\
\text{R0} & = 1 \\
\text{R1} & = y \\
\text{for } i=0 \text{ to } n-1 \text{ do} \\
\text{if } x_i = 0 \text{ then} \\
\text{R1} & = \text{R0} \ast \text{R1} \mod N \\
\text{R0} & = \text{R0} \ast \text{R0} \mod N \\
\text{else} \\
\text{R0} & = \text{R0} \ast \text{R1} \mod N \\
\text{R1} & = \text{R1} \ast \text{R1} \mod N \\
\text{return } & \text{R0} \\
\text{\}}
\end{align*}
\]

\( c_b=0 \) and \( c_b=1 \) take the same time

Modular multiplications done with Montgomery multiplier
Montgomery Multiplication

• Montgomery multiplication changes $mod\ q$ operations to $mod\ 2^k$
  – This is much faster (since $mod\ 2^k$ is achieved taking the last $k$ bits)

• Computing $c \equiv a*b \ mod\ q$ using Montgomery multiplication
  1. For the given $q$, select $R=2^k$ such ($R > q$) and $gcd(R,q) = 1$
  2. Using Extended Euclidean Algorithm find two integers to compute $R^{-1}$ and $q'$ such that $R.R^{-1} - q.q' = 1$
  3. Convert multiplicands to their Montgomery domain:
     \begin{align*}
     A & \equiv aR \ mod\ q \\
     B & \equiv bR \ mod\ q
     \end{align*}
  4. Compute $abR \ mod\ N$ using the following steps
     \begin{align*}
     S &= A * B \\
     S &= S + (S * q' \ mod\ R) * q / R \\
     \text{if} \ (S > q) \\
     S &= S - q \\
     \text{return} \ S
     \end{align*}
     Requires 3 integer multiplications
  5. Perform $S*R^{-1} \ mod\ q$ to obtain $ab \ mod\ q$

http://www.hackersdelight.org/MontgomeryMultiplication.pdf
Montgomery Multiplier in the Montgomery Ladder

**Input:** c, y  
**Output:** $y^c \mod N$

```plaintext
exp(c, y) {
    R0 = 1 * R \mod N
    R1 = y * R \mod N

    for i=0 to n-1 do
        if ci = 0 then
            R1 = R0 * R1
            R0 = R0 * R0
        else
            R0 = R0 * R1
            R1 = R1 * R1

    return (R0 * R^{-1})
}
```

- Convert to Montgomery domain.
- Multiplications in Montgomery domain. Note. Each result is also in Montgomery domain.
- Return to Original domain
The final ‘if’ in Montgomery Multiplication

- Observation
  
  \[ \text{Pr[ExtraReduction]} = \frac{y \mod q}{2R} \]

- Consider \( y \) to be an integer increasing in value
- As \( y \) approaches \( q \), \( \text{Pr[ExtraReduction]} \) increases
- When \( y \) is a multiple of \( q \), \( \text{Pr[ExtraReduction]} \) drops
- Extra reductions causes execution time to increase

Extra reduction step

\[ S = (A \times B) R^{-1} \mod q \]

If \( S > q \) then \( S = S - q \)

\( g = y \) is the ciphertext in the plots
Another timing variation due to Integer multiplications

• 30-40% of OpenSSL RSA decryption execution time is spent on integer multiplication

• If multiplicands have the same number of words $n$, OpenSSL uses Karatsuba multiplication $O(n^{\log_2 3})$

• If integers have unequal number of words $n$ and $m$, OpenSSL uses normal multiplication $O(nm)$

these further cause timing variations...
Summary of Timing Variations

<table>
<thead>
<tr>
<th></th>
<th>$y &lt; q$</th>
<th>$y &gt; q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Montgomery Effect</td>
<td>Longer</td>
<td>Shorter</td>
</tr>
<tr>
<td>Multiplication Effect</td>
<td>Shorter</td>
<td>Longer</td>
</tr>
</tbody>
</table>

Opposite effects, but one will always dominate.
Retrieving a bit of q

Assume the attacker has the top i-1 bits of q,
High level attack to get the \( i^{th} \) bit of q

1. Set \( y_0 = (q_{i-1}, q_{i-2}, q_{i-3}, \ldots q_{i-i-1}, 0, 0, \cdots) \)
   Set \( y_1 = (q_{i-1}, q_{i-2}, q_{i-3}, \ldots q_{i-i-1}, 1, 0, 0, \cdots) \)

   note that
   if \( q_i = 0, \quad y_0 \leq q < y_1 \)
   if \( q_i = 1, \quad y_0 < y_1 \leq q \)

2. Sample decryption time for \( y_0 \) and \( y_1 \)
   \( t_0 : DecryptionTime(y_0) \)
   \( t_1 : DecryptionTime(y_1) \)

3. If \( |t_1 - t_0| \) is large \( \rightarrow q_i = 0 \) (corresponds to \( y_0 \leq q < y_1 \))
   else \( q_i = 1 \) (corresponds to \( y_0 < y_1 \leq q \))
What’s happening here?

Assume Montgomery multiplier dominates over Integer multiplication

• **Case 1 :** $t_1$
  
  $y_0 < y_1 \leq q$
What’s happening here?

Assume Montgomery multiplier dominates over Integer multiplication

- **Case 2 :** $t_0$  
  $y_0 < q \leq y_1$

Due to Montgomery

```
\begin{align*}
\text{Decryption time} & \\
y_0 \text{ case} & \\
y_1 \text{ case} & \\
kq & \\
\text{value of } y & 
\end{align*}
```
What’s happening here?

Assume Montgomery multiplier dominates over Integer multiplication

- **Case 2**: $t_0 \quad y_0 < q \leq y_1$

  Due to Montgomery

What happens when integer multiplier dominates or Montgomery multiplier?
How does this work with SSL?

How do we get the server to decrypt our y?
Normal SSL Session Startup

1. ClientHello
2. ServerHello (send public key)
3. ClientKeyExchange \((r^e \mod N)\)

Result: Encrypted with computed shared master secret
Attacking Session Startup

1. ClientHello
2. ServerHello (send public key)
3. Record time $t_{start}$
   Send guess $y_0$ or $y_1$
4. Alert
5. Record time $t_{end}$
   Compute $t_{start} - t_{end}$

Attack Client

USENIX SSL Server
Timing Attacks on Block Ciphers

Cache Attacks and Countermeasures: the Case of AES

Cache Timing Attacks on AES
https://cr.yp.to/antiforgery/cachetiming-20050414.pdf
Block Cipher Constructions

- Sboxes typically implemented with look up tables

- If block cipher is implemented in a system with cache memory, then the look up tables present could lead to timing attacks
Memory Hierarchies in Systems

- **Von-Neumann bottleneck**
  - Due to high speed of processors and relatively low speed of RAM

- **Goal of Memory Hierarchy**
  - Low latency, high bandwidth, high capacity, low cost

![Diagram of Memory Hierarchy](image)
Cache Memories

Memory Load Instruction{

    If data present in L1 cache (L1 cache hit) {
        then return data from L1 cache
    } else if data not present in L1 cache (L1 cache miss) {
        if data present in L2 cache (L2 cache hit) {
            return data from L2 cache and fill L1 cache
        } else if data present in L3 cache (L3 cache hit) {
            return data from L3 cache and fill L1 and L2 caches
        } else {
            read data from RAM and fill in all caches
        }
    }
}
Address Mapping of Cache Memories

- Memory divided into blocks
  One block typically 64 bytes
- Cache memory divided into lines.
  Line size = block size.
- There is a mapping from blocks in memory to lines in the cache
  - Example direct mapped cache.
    - If the cache size contains 4 lines, then every 4-th block gets mapped to the same cache line
Address Mapping of Cache Memories

• Cache Details:
  – Let the number of words in a cache line be $2^\delta$
  – Let the number of lines in the cache be $2^b$
  – The number of words in the cache is therefore $2^{b+\delta}$

• How to compute the mapping?

![Diagram showing address mapping](attachment://address_mapping_diagram.png)

- 32 – (b+\delta) bits (Tag bits)
- b bits (Line Address)
- \(\delta\) bits (Word Address)

Mapping a 32 bit address
Organization of a Direct Mapped Cache

\[
\text{const unsigned char } \text{T0[256]} = \{ 0x63, 0x7C, 0x77, 0x7B, \ldots \};
\]

T0 address is 0x804af60
Access Driven Attacks

- Assumptions
  - The attacker shares the same hardware as the victim. For instance, cloud infrastructure.
  - The attacker manipulates the system in such a way as to track execution patterns of a victim process.
  - These execution patterns are used to infer sensitive data about the victim.
S-boxes and Cache Memories

S-boxes generally implemented as lookup tables. Arrays stored in memory.

When accessed, a part of the table gets loaded into the cache memory.

Subsequent accesses to the part of the table results in cache hits (unless evicted).
If I know the index into the table \((I_0)\) and I know \(P_0\) then \(P_0 \text{xor} K_0 = I_0\).

Thus, \(P_0 \text{xor} I_0 = K_0\).

We will see how few bits of \(I_0\) can be recovered from monitoring the execution time of the cipher.
Cache State when a cipher is executed

Changing plaintext or key will alter how the cache memory is used.

- Cache line not used during the cipher execution
- Cache line filled up by the cipher execution
Cache State when a cipher is executed

Pt1, Pt2, Pt3 are same in one byte. All other bytes may be different

Cipher(Pt1, Key1)  Cipher(Pt1, Key1)  Cipher(Pt3, Key1)

When plaintexts have one byte which is same, then there exists one cache line that is filled in every encryption

- Cache line filled in every encryption
- Cache line not used by the cipher execution
- Cache line filled up by the cipher execution
Evict+Time Attack

Repeat multiple times

1. P is a randomly chosen plain text (with one byte say P0 fixed)
2. Invoke encryption of P
3. Evict a random line in the cache (say line L)
4. Invoke encryption of P (again) and time encryption

- Note that encryption of P occurs twice. So the second encryption will predominantly result in cache hits.
- If line L is used during the encryption, a cache miss arises... leading to an increase in execution time of 2\textsuperscript{nd} encryption
- If line L is not used during the encryption, no additional cache miss arises .... There may not be a significant increase in the execution time of 2\textsuperscript{nd} encryption
What’s Happening here?

Three scenarios arise

1. Evicted line L (Red) collides with the yellow

2. Evicted line (Red) collides with the brown. But this is unlikely to happen for every encryption, since P changes

3. Evicted line (Red) does not collide with Yellow or Brown. This is also unlikely to happen in every encryption, since P changes.

Evicted line, Picked randomly is Shown in red
What’s Happening here?

Three scenarios arise

1. Evicted line L (Red) collides with the yellow

2. Evicted line (Red) collides with the brown. But this is unlikely to happen for every encryption, since P changes

3. Evicted line (Red) does not collide with Yellow or Brown. This is also unlikely to happen in every encryption, since P changes.

What can we infer?

In case 1, there is always an additional Cache miss during the second encryption.

In case 2 or 3, an additional cache miss may or may not occur.

Thus avg time in case 1 > avg time in case 2 or 3
Prime+Probe

- Uses a spy program to determine cache behavior
Limitations

• Number of bits recovered is restricted by the cache line size.

• Solved to certain extent by targeting cache hits in the second round of the block cipher
Bernstein’s Profiled Time Driven Cache Attacks
Time Profiles

The table is accessed at location $P_0 \wedge K_0$.

Each value of $(P_0 \wedge K_0)$ results in a unique timing distribution.
The table is accessed at location $P_0 \wedge K_0$.

Each value of $(P_0 \wedge K_0)$ results in a unique timing distribution.
Bernstein’s Cache Timing Attack

1. Put key to all ZEROs and perform experiment

2. Repeat experiment with unknown key

3. Correlate the two results
Results for the Block Cipher AES

<table>
<thead>
<tr>
<th>key</th>
<th>Correct key</th>
<th>Ten most likely keys for each byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_0^{(0)}$</td>
<td>11</td>
<td>4e 47 41 4a 46 4c 48 45 4f 44</td>
</tr>
<tr>
<td>$k_1^{(0)}$</td>
<td>22</td>
<td>0e 22 c2 2f ca 33 e1 06 23 c9</td>
</tr>
<tr>
<td>$k_2^{(0)}$</td>
<td>33</td>
<td>33 38 3b 3a 34 37 39 0c 3f a7</td>
</tr>
<tr>
<td>$k_3^{(0)}$</td>
<td>44</td>
<td>83 89 8a 81 41 8b 84 46 4b 4a</td>
</tr>
<tr>
<td>$k_4^{(0)}$</td>
<td>55</td>
<td>d1 de d9 a8 d0 d3 aa a5 a0 a1</td>
</tr>
<tr>
<td>$k_5^{(0)}$</td>
<td>66</td>
<td>8f 52 c3 7a 2b 50 1a 23 f6 4a</td>
</tr>
<tr>
<td>$k_6^{(0)}$</td>
<td>77</td>
<td>79 73 78 74 77 7e 7f 75 8d 8e</td>
</tr>
<tr>
<td>$k_7^{(0)}$</td>
<td>88</td>
<td>8e 87 8f 80 8a 86 89 8d 8b 88</td>
</tr>
<tr>
<td>$k_8^{(0)}$</td>
<td>99</td>
<td>99 93 98 90 ba 1e 7a af 70 13</td>
</tr>
<tr>
<td>$k_9^{(0)}$</td>
<td>aa</td>
<td>b4 e2 7b e8 b1 c8 53 7a 79 bb</td>
</tr>
<tr>
<td>$k_{10}^{(0)}$</td>
<td>bb</td>
<td>65 57 5f b2 24 b6 60 25 5e 80</td>
</tr>
<tr>
<td>$k_{11}^{(0)}$</td>
<td>cc</td>
<td>c6 c2 ce ca cb cc c1 c0 14 cf</td>
</tr>
<tr>
<td>$k_{12}^{(0)}$</td>
<td>dd</td>
<td>53 5b 50 52 49 58 5d 51 d1 48</td>
</tr>
<tr>
<td>$k_{13}^{(0)}$</td>
<td>ee</td>
<td>7c e0 4e 98 94 eb e5 d7 b3 3b</td>
</tr>
<tr>
<td>$k_{14}^{(0)}$</td>
<td>ff</td>
<td>ea fd fb 3a e1 a4 e9 03 f1 ff</td>
</tr>
<tr>
<td>$k_{15}^{(0)}$</td>
<td>00</td>
<td>05 01 06 02 04 08 03 0a 00 0c</td>
</tr>
</tbody>
</table>
Results for the Block Cipher CLEFIA

<table>
<thead>
<tr>
<th>key</th>
<th>Correct key</th>
<th>Ten most likely keys for each byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>RK00</td>
<td>f4</td>
<td>f4, e2, c1, eb, 52, 18, e1, d7, 14, 44</td>
</tr>
<tr>
<td>RK01</td>
<td>d0</td>
<td>d0, 52, f0, df, 46, 51, d8, 44, f2, d7</td>
</tr>
<tr>
<td>RK03</td>
<td>6a</td>
<td>6a, 5f, 94, 92, e8, 48, 6c, 75, a9, b6</td>
</tr>
<tr>
<td>RK10</td>
<td>ca</td>
<td>ca, a7, 5b, 40, 54, 52, bf, 58, 51, 53</td>
</tr>
<tr>
<td>RK11</td>
<td>7b</td>
<td>7b, 46, db, d1, c6, c4, 52, 56, 8f, 79</td>
</tr>
<tr>
<td>RK12</td>
<td>91</td>
<td>91, 13, 5a, 8c, f2, 14, 64, a8, f6, 36</td>
</tr>
<tr>
<td>RK13</td>
<td>60</td>
<td>60, ab, 07, 68, c5, ec, 9c, 78, e9, 16</td>
</tr>
<tr>
<td>RK20 + WK00</td>
<td>fe</td>
<td>fe, f8, 00, 06, ec, 14, 11, 1c, f6, 1b</td>
</tr>
<tr>
<td>RK21 + WK01</td>
<td>57</td>
<td>57, 51, 62, a7, 5a, f7, 64, 24, e1, 9f</td>
</tr>
<tr>
<td>RK22 + WK02</td>
<td>3c</td>
<td>3c, ea, c5, eb, 3d, 8c, be, 92, 11, ec</td>
</tr>
<tr>
<td>RK23 + WK03</td>
<td>80</td>
<td>80, 51, 02, 58, 57, 3c, d8, 89, 10, 74</td>
</tr>
<tr>
<td>RK30 + WK10</td>
<td>6b</td>
<td>6b, 7b, 42, 90, 6f, a3, 56, d6, 3d, a9</td>
</tr>
<tr>
<td>RK31 + WK11</td>
<td>40</td>
<td>40, 4a, b1, 88, fb, 92, 16, 2b, 05, 13</td>
</tr>
<tr>
<td>RK32 + WK12</td>
<td>16</td>
<td>16, 05, 94, fb, 45, 6b, b9, 15, f8, 6e</td>
</tr>
<tr>
<td>RK33 + WK13</td>
<td>36</td>
<td>36, f2, 42, a8, ad, 86, 80, c5, 1b, 34</td>
</tr>
<tr>
<td>RK40</td>
<td>7e</td>
<td>7e, e0, fe, e8, 01, 11, ff, 07, 1c, 12</td>
</tr>
<tr>
<td>RK41</td>
<td>32</td>
<td>32, 2f, 34, 26, 38, 31, 35, 3f, 3e, 29</td>
</tr>
<tr>
<td>RK42</td>
<td>50</td>
<td>50, 5d, 00, a0, 81, f0, 65, 82, b0, 03</td>
</tr>
<tr>
<td>RK43</td>
<td>e1</td>
<td>e1, 0e, 37, dc, 63, cc, c8, e5, 89, 77</td>
</tr>
<tr>
<td>RK50</td>
<td>eb</td>
<td>eb, 9b, da, 85, 1e, f8, 3e, fe, 4c, 99</td>
</tr>
<tr>
<td>RK51</td>
<td>11</td>
<td>11, 24, e9, ef, 33, 93, cd, 0e, d2, 17</td>
</tr>
<tr>
<td>RK52</td>
<td>47</td>
<td>47, 37, 92, f8, 99, 8c, bb, 34, b2, 52</td>
</tr>
<tr>
<td>RK53</td>
<td>35</td>
<td>35, b7, 38, 7f, e7, 5f, 31, e8, 8b, ed</td>
</tr>
</tbody>
</table>
Countermeasures for Timing Attacks

• Requirements for a successful Side Channel Attack
  – Perturbations:
    • When the cipher executes, some entity in the system must be disturbed (perturbed)
  – Manifestations:
    • These perturbations should be manifested through some channel (for instance a power glitch)
  – Observables:
    • The manifestations should be observable / measurable in spite of all the noise
• Preventing any one of these requirements can counter side channel attacks.
Preventing Cache Timing Attacks

- Adding noise during the encryption ....
- Constant time implementations ..... difficult
- Non-cached memory access
- Specialized cache designs
  - Partitioned cache
  - Random permutation cache
- Specialized Instructions
- Prefetching
- Fuzzing Clocks
  - Virtual time stamp counters
Fault Attacks

Fault Attacks

- Active Attacks based on induction of faults
- First conceived in 1996 by Boneh, Demillo and Lipton
- E. Biham developed Differential Fault Analysis (DFA) attacker DES
Illustration of a Fault Attack

PLAIN TEXT

ENCRIPTION

FAULT FREE CIPHER TEXT

FAULT INDUCTION

ENCRIPTION

FAULTY CIPHER TEXT

ANALYSIS
How to achieve fault injection

- Laser
- Clock Glitching
- Power Glitching
- Temperature???
Fault Injection Using Clock Glitches

An Internal state of The AES on logic scope
Fault Models

- **Bit model**: When fault is injected, exactly one bit in the state is altered
  
  eg. 8823124345 → 8833124345

- **Byte model**: exactly one byte in the state is altered
  
  eg. 8823124345 → 8836124345

- **Multiple byte model**: faults affect more than one byte
  
  eg. 8823124345 → 8836124333

Fault injection is difficult.... The attacker would want to reduce the number of faults to be injected
Fault Attack on RSA

RSA decryption has the following operation

\[ x = y^a \mod n \]

where \( a \) is the private key \( y \) the ciphertext and \( x \) the plain text.

Suppose, the attacker can inject a fault in the \( i^{th} \) bit of \( a \). Thus she would get two ciphertexts:

The fault free ciphertext \( x = y^a \mod n \)

The faulty ciphertext \( \tilde{x} = y^{\tilde{a}} \mod n \)
Fault Attack on RSA

$a$ and $\tilde{a}$ differ by exactly 1 bit; the $i^{th}$ bit. Thus

$$a - \tilde{a} = \begin{cases} 
2^i & \text{if } a_i = 1 \\
-2^i & \text{if } a_i = 0 
\end{cases}$$

Now consider the ratio

$$\frac{x}{\tilde{x}} = \frac{y^a}{y^{\tilde{a}}} \mod n = y^{a-\tilde{a}} \mod n$$

Thus, 

$$\frac{x}{\tilde{x}} = \begin{cases}
 y^{2^i} & \text{if } a_i = 1 \\
 y^{-2^i} & \text{if } a_i = 0
\end{cases}$$

The attacker thus gets 1 bit of $a_i$. Similar faults on other bits will reveal more information about the private key $a_i$. 

75
What a fault does to a block cipher?

- A fault (generally at the s-box input) creates a difference wrt the fault free encryption.
- This difference is propagated and diffused to multiple output bytes of the cipher.
- The attacker thus has 2 ciphertexts: (1) the fault free ciphertext ($C$) and (2) the faulty ciphertext ($C^*$).
A Simple Fault Attack on AES

• Let’s assume that the attacker has the capability of resetting a particular line during the AES round key addition. (i.e. exactly one bit is reset)

• Attack Procedure
  1. Put plaintext to 0s and get ciphertext C
  2. Put plaintext to 0s. Inject fault in the ith bit as shown. Get the ciphertext C*
  3. If C=C*, we infer $K_i = 1$
     If $C \neq C^*$, we infer $K_i = 0$

• This techniques requires 128 faults to be injected.
  – difficult,,,, can we do better?
Differential Fault Attack on AES

• Differential characteristics of the AES s-box
DFA on last round of AES (using a single bit fault)

\[ C_0 + C_0^* = S(p) + S(p+f) \]

Since it is a single bit fault, f can take on one of 7 different values:
(00000001), (00000010), (00000100), (00001000), ..., (10000000)

The above equation on average will have around 8 different solutions for p. Each value of p would give a candidate for k. Thus, there are 8 key candidates.
DFA on last round of AES (using a single bit fault)

- Each bit fault results in 8 potential key values for the byte
- There are 16 key bytes. Thus 16 faults need to be injected.
- In total key space reduces from $2^{128}$ to $8^{16}$ (ie. $2^{48}$)
  - A key space search of $2^{48}$ do-able in reasonable time
DFA on 9\textsuperscript{th} Round of AES (fault in a byte)

- Fault injected after s-box operation in the 9\textsuperscript{th} round.
- It is a byte level fault, thus, the fault ‘f’ can take on any of 256 values (0, 1, 2, ..., 255)
- Due to the mix-column, 4 difference equations can be derived

\[
\begin{align*}
2f &= S^{-1}(C_{0,0} \oplus K_{0,0}^{10}) \oplus S^{-1}(C_{0,0}^{*} \oplus K_{0,0}^{10}) \\
f &= S^{-1}(C_{1,3} \oplus K_{1,3}^{10}) \oplus S^{-1}(C_{1,3}^{*} \oplus K_{1,3}^{10}) \\
f &= S^{-1}(C_{2,2} \oplus K_{2,2}^{10}) \oplus S^{-1}(C_{2,2}^{*} \oplus K_{2,2}^{10}) \\
3f &= S^{-1}(C_{3,1} \oplus K_{3,1}^{10}) \oplus S^{-1}(C_{3,1}^{*} \oplus K_{3,1}^{10})
\end{align*}
\]
Solving the Difference Equations

Each equation has the form: \( A = B \oplus C \)

where, A, B, C are of 8 bits each.

For a uniformly random choice of A, B, and C, the probability that the above equation is satisfied is \((1/2^8)\). The maximum space of (A,B,C) is \(2^{24}\). Of these values, \(2^{16}\) will satisfy the above equation.
Solving the Difference Equations

Each equation has the form: \[ A = B \oplus C \]
where, \( A, B, C \) are of 8 bits each.

For a uniformly random choice of \( A, B, \) and \( C \),
the probability that the above equation is satisfied is \( \frac{1}{2^{8}} \).

The maximum space of \( (A,B,C) \) is \( 2^{24} \). Of these values, \( 2^{16} \) will satisfy the above equation.

In our case, there are 5 unknowns (4 keys and \( f \)) and 4 equations.
For uniformly random chosen values of the 5 unknowns, the probability that all 4 equations are satisfied is \( p = \frac{1}{2^{8}}^{4} \).

The space reduction for the 5 variables is therefore from \( p(2^{8})^{5} = 2^{8(5-4)} = 2^{8} \).

The key space is \( 2^{32} \). From the above, it has reduced to just \( 2^{8} \).

Each fault reveals 32 bits of the 10th round key.
Thus 4 faults are required to reveal all 128 key bits. The offline search space is \( 2^{32} \).
Can we do better?
DFA on AES with a single fault

- As mentioned previously, 4 faults are required in the 9\textsuperscript{th} round to reveal the entire key.
- Instead of the 9\textsuperscript{th} round, suppose we inject the fault in the 8\textsuperscript{th} round.
DFA on AES in the 8\textsuperscript{th} round

- A single fault injected in the 8\textsuperscript{th} round will spread to 4 bytes in the 9\textsuperscript{th} round.
- This is equivalent to having 4 faults in each of the 4 columns.
- A single fault can thus be used to determine all key bytes.
- The offline key space is $2^{32}$ as before