Side Channel Attacks

Chenglu Jin
Department of Electrical & Computer Engineering
University of Connecticut
Email: chenglu.jin@.uconn.edu

Based on and extracted from Nickolai Zeldovitch, Computer System Security, course material at http://css.csail.mit.edu/6.858/2014/

With help from Marten van Dijk
1. Introduction
   1. Timing channel
   2. Power channel
   3. EM radiation channel
   4. Acoustic channel
   5. Photonic emission channel

2. Timing attack on RSA
   1. Background
   2. Target RSA implementation
   3. Attack
   4. Defense
   5. Demo

3. Cache timing attack
   1. Attacks
   2. Countermeasures
   3. Demo

4. Differential Power Analysis
   1. Differential Power Analysis on AES
In cryptography, a side-channel attack is an attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms (compare cryptanalysis).

1. Timing channel
2. Power channel
3. EM radiation channel
4. Acoustic channel
5. Photonic Emission channel

https://en.wikipedia.org/wiki/Side-channel_attack
Timing Side Channel

The computation time depends on the value of secret data, so one can uncover the secret by timing the execution of a particular operation.

**FIGURE 1:** RSAREF Modular Multiplication Times

Different secret data can lead to different data access pattern (cache hit or cache miss), and cache hit and miss has a huge timing difference. Therefore, one can extract the secret by observing the access time of each cache access.

Daniel J. Bernstein. “Cache-timing attacks on AES”. 2005
The power consumption of a chip depends on the secret data that is computing on the chip. One is able to uncover the secret data by measuring the power consumption of the entire chip.

**Figure 1:** SPA trace showing an entire DES operation.

EM radiation depends on the secret data that is being processed.

Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic Analysis: Concrete Results". CHES'01
Acoustic Channel

Acoustic frequency from different motherboard components leak information about the instructions performed by the target’s CPU.

Figure 7: Acoustic measurement frequency spectrogram of a recording of different CPU operations using the Bruel&Kjær 4939 microphone capsule. The horizontal axis is frequency (0-310 kHz), the vertical axis is time (3.7 sec), and intensity is proportional to the instantaneous energy in that frequency band.
Photonic Emission Channel

Fig. 2. 120 s emission images of memory accesses to two adjacent memory rows obtained with the Si-CCD detector.

Photonic emission pattern is data dependent, so it can also be used to extract the secret data.
Outline

1. Introduction
   1. Timing channel
   2. Power channel
   3. EM radiation channel
   4. Acoustic channel
   5. Photonic emission channel

2. Timing attack on RSA
   1. Background
   2. Target RSA implementation
   3. Attack
   4. Defense
   5. Demo

3. Cache timing attack
   1. Attacks
   2. Countermeasures
   3. Demo

4. Differential Power Analysis
   1. Differential Power Analysis on AES
Timing Attack on RSA

- This paper demonstrates an attack to reconstruct private key of RSA over the network.

David Brumley and Dan Boneh, “Remote timing attacks are practical”. Computer Networks’05.
RSA Background

- RSA: parameters

- 1. Pick two random primes, p and q. Let \( n = p \times q \). A reasonable key length, i.e., \( |n| \), is 2048 bits today.

- 2. Euler's function \( \phi(n) = (p-1) \times (q-1) \)
  - For all \( a \) and \( n \), \( a^{\phi(n)} = 1 \mod n \)

- Encryption: \( c = m^e \mod n \)

- Decryption: \( m = c^d \mod n \)

- e is public key and d is private key, such that \( m^{e \times d} \mod n = m \)

- By using \( \phi(n) \) function and extended Euclidean algorithm, we can easily compute d from e.

Problems of Plain RSA

- Ciphertexts are multiplicative
  - \( E(a) \cdot E(b) = a^e \cdot b^e = (ab)^e \)

- RSA is deterministic encryption
  - Ciphertext of the same plaintexts are the same.

- Solution:
  - Padding: take plaintext message bits, add padding bits before and after plaintext. Padding bits introduce randomness into encryption.

Bellare M, Rogaway P. Optimal asymmetric encryption EUROCRYPT'94
Optimal Asymmetric Encryption Padding

a.k.a. OAEP

To encode,
1. messages are padded with \( k_1 \) zeros to be \( n - k_0 \) bits in length.
2. \( r \) is a randomly generated \( k_0 \)-bit string
3. \( G \) expands the \( k_0 \) bits of \( r \) to \( n - k_0 \) bits.
   \[ X = m00..0 \oplus G(r) \]
4. \( H \) reduces the \( n - k_0 \) bits of \( X \) to \( k_0 \) bits.
   \[ Y = r \oplus H(X) \]
5. The output is \( X \parallel Y \) where \( X \) is shown in the diagram as the leftmost block and \( Y \) as the rightmost block.

To decode,
1. recover the random string as \( r = Y \oplus H(X) \)
2. recover the message as \( m00..0 = X \oplus G(r) \)

RSA implementation

- Key problem: fast modular exponentiation.
  - In general, quadratic complexity.
  - Multiplying two 1024-bit number is slow
  - Computing the modulus for 1024-bit numbers is slow. (1024-bit division).
How to do modular exponentiation of a large number efficiently?

Short answer: split it into two smaller numbers

Chinese Remainder Theorem:

First, Compute \( m_1 = c^d \mod p \), and \( m_2 = c^d \mod q \).

Then, Compute \( m = q \cdot c_p \cdot m_1 + p \cdot c_q \cdot m_2 \mod n \)
- Where \( c_p = q^{-1} \mod p \), \( c_q = p^{-1} \mod q \)

It has 2x speedup.
- Shorter modular exponentiation in the first step
- Only modular multiplication and addition in second step

Preneel, Bart and Paar, Christof and Pelzl, Jan. "Understanding cryptography: a textbook for students and practitioners". Springer 2009
How to do modular exponentiation efficiently?

Short answer: repeated squaring

Example: we want to compute $a^{16}$

1. Do 15 multiplications

2. Do 4 squaring $((((a^2)^2)^2)^2)) = a^8$
Optimization 2

- Repeated squaring and Sliding windows

Algorithm 1 Multiply and Square Algorithm

To compute $g^K$

```plaintext
1: procedure Mul - Squ(g, K)
2:   Convert K into binary representation $k_0, k_1, \ldots, k_n$, where $k_0 = 1$
3:   if $K == 0$ then
4:     Result = 1
5:     return Result
6:   else
7:     Result = g
8:     for $i \gets 1, n$
9:       if $k_i == 1$ then
10:          Result = $M(\text{Result, Result})$
11:       else
12:          Result = $M(\text{Result, g})$
13:       end if
14:     end for
15:     return Result
16: end procedure
```

If we consider more than one consecutive bits in $k$ in each iteration, we call it sliding window. e.g. if $k_ik_{i+1} = 3$, then square twice and multiply with $g^3$. 
How to do modular operation efficiently?

Short answer: avoid division, only use multiplication and subtraction

**Montgomery representation:** multiply everything by some factor \( R \).

\[
\begin{align*}
a \mod q & \leftrightarrow aR \mod q \\
b \mod q & \leftrightarrow bR \mod q \\
c = a \times b \mod q & \leftrightarrow cR \mod q = (aR \times bR)/R \mod q = \\
& \quad (aR \mod q) \times (bR \mod q) \times R^{-1} \mod q.
\end{align*}
\]

Additional division by \( R \) should be very cheap, either shifting or multiplying with precomputed \( R^{-1} \). (E.g. \( R = 2^n \))

Example:

\( N = 17, R = 100, R^{-1} = 8. \) The Montgomery forms of 3, 5, 7, and 15 are 300 mod 17 = 11, 500 mod 17 = 7, 700 mod 17 = 3, and 1500 mod 17 = 4.

Montgomery forms of 7 and 15 modulo 17 is the product of 3 and 4, which is 12.

\[12 \times R^{-1} \mod N = 12 \times 8 \mod 17 = 11 \text{ (Montgomery form of 3)}\]

https://en.wikipedia.org/wiki/Montgomery_modular_multiplication
One remaining problem: result \((aR \times bR) / R\) will be \(< R\), but might be \(> q\).
- Requires subtraction of \(q\). This is called extra reduction.
- \(Pr[\text{extra reduction}] = (x \mod q) / 2R\), when we compute \(x^d \mod q\)

Notice: If extra reduction happens, the computation costs more time. This timing leaks information.
How to do multiplication efficiently?

Short answer: select an efficient multiplier on the fly

Two options: pair-wise multiplier and Karatsuba multiplier

First, split two 512-bit numbers into 32-bit components.

Second, select one multiplication from two different multiplications: pair-wise multiplication vs Karatsuba multiplication

Pair-wise:
- Requires $O(nm)$ time if two numbers have $n$ and $m$ components respectively
- $O(n^2)$ if the two numbers are close

Karatsuba:
- Requires $O(n^{1.585})$ time

In the implementation, the software selects the most efficient multiplication to compute according to the values of $n$ and $m$.

Notice: selection of multipliers leaks information.
The big picture of RSA Decryption

\[ c_0 = c \mod q \quad c'_0 = c_0 \cdot R \mod q \quad m'_0 = (c'_0)^d \mod q \]

- Use sliding window for bits of the exponent \( d \)
- Karatsuba if \( c'_0 \) and \( q \) have the same number of 32-bit parts
- Extra reductions proportional to \((c'_0)^z \mod q) / 2R; z\) comes from sliding window

\[ ... \]
Construction of attack vectors

- Let $q = q_0 \ldots q_N$, where $N = |q|$
- Assume we know some number $j$ of high-order bits of $q$ ($q_0$ to $q_j$)
- Construct two approximations of $q$, guessing $q_{j+1}$ is either 0 or 1:
  - $g = q_0 q_1 \ldots q_j \ 0 \ 0 \ldots 0 \ 0$
  - $g_{hi} = q_0 q_1 \ldots q_j \ 1 \ 0 \ldots 0 \ 0$
- Trigger the decryption $g^d$ and $g_{hi}^d$. (Padding is checked after decryption)
- Two cases:
  - $q_{j+1} = 0 \Rightarrow g < q < g_{hi}$: time($g^d$) and time($g_{hi}^d$) have noticeably difference
    - $g_{hi}$ mod $q$ is small
    - Less time: fewer extra reductions
    - More time: switch from Karatsuba to pair-wise multiplication
  - $q_{j+1} = 1 \Rightarrow g < g_{hi} < q$: time($g^d$) and time($g_{hi}^d$) have no much difference
Evaluation

Zero-one gap ($T_g - T_{g_{hi}}$) for three different keys

Effect of extra reduction.
What if the two effects are canceled out?

Zero-one gap ($T_g - T_{g_{hi}}$) for three different keys
For every bit of $g$ we measure the decryption time for a neighborhood of values $g; g+1; g+2; \ldots; g+n$. We denote this neighborhood size by $n$. 
Effect of increased neigh. size

- Increasing neigh. = larger zero-one gap

![Graph showing time difference in CPU cycles against bits guessed of factor q for different neighborhood sizes.](image)
Countermeasures

- **RSA blinding**
  - Choose random \( r \) when decryption
  - Randomize \( c' = c \times r^e \mod n \)
  - Multiplicative property of RSA => the decrypted result is \( m' = m \times r \)
  - \( m = m' / r \)

- **Constant execution time**
  - Montgomery Ladder

- **Disallow the access to the precise timer**
  - Attacker may still be able to figure out the information from throughput.

\[ x_1 = x; \quad x_2 = x^2 \]

for \( i = k - 2 \) to 0 do

If \( n_i = 0 \) then

\[ x_2 = x_1 \times x_2; \quad x_1 = x_1^2 \]

else

\[ x_1 = x_1 \times x_2; \quad x_2 = x_2^2 \]

return \( x_1 \)

https://en.wikipedia.org/wiki/Montgomery_modular_multiplication
Demo

- For demo purpose:
- \( p = 97, q = 103, e = 31 \). \( N = p \times q = 92391 \)
- Private key: \( d = 7 \)

https://github.com/stoutbeard/crypto
1. Introduction
   1. Timing channel
   2. Power channel
   3. EM radiation channel
   4. Acoustic channel
   5. Photonic emission channel

2. Timing attack on RSA
   1. Background
   2. Target RSA implementation
   3. Attack
   4. Defense
   5. Demo

3. Cache timing attack
   1. Attacks
   2. Countermeasures
   3. Demo

4. Differential Power Analysis
   1. Differential Power Analysis on AES
Cache side channel attacks

- Data present in caches can be accessed faster than from memory
- For multilevel caches, data accessed from L1 cache has lower latency than from an L2 cache
- The cache interference and time difference for the access patterns leaks information:
  - Certain memory contents exist in cache or not
  - Shows that data has been accessed recently
- This attack is useful to find keys for encryption process
The attacker wants to know if 0x1000, which maps to cache Set 1, was accessed
• He triggers the encryption and times it.
• He evicts everything from Set1.
• He runs the encryption again and times it.
• It takes longer than step 1, he knows that the encryption process accessed 0x1000.

The attacker wants to know if 0x4000, which maps to cache Set 4, was accessed
• He triggers the encryption and times it.
• He evicts everything from Set4.
• He runs the encryption again and times it.
• It takes roughly the same time, he knows that the encryption process didn’t access 0x4000.

Prime + probe technique consists of 3 stages

- Prime stage: The attacker fills the cache with his own cache lines.
- Victim accessing stage: The victim process runs
- Probing stage: The attacker accesses the priming data again. If the victim process evicts the primed data, the reloading will incur cache miss.

The attacker wants to know if a particular address in cache **Set 1** was accessed

- He fills **Set 1** with his data.
- He runs the victim process.
- He reloads all his data in **Set 1**.
- It takes longer, he knows that the victim process accessed **Set 1**.

The attacker wants to know if a particular address in cache **Set 4** was accessed

- He fills **Set 4** with his data.
- He runs the victim process.
- He reloads all his data in **Set 4**.
- It takes lesser time, he knows that the victim process didn’t access **Set 4**.

---

**Prime + probe technique**

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>hit</td>
<td>hit</td>
<td>miss</td>
<td>hit</td>
</tr>
<tr>
<td>hit</td>
<td>hit</td>
<td>hit</td>
<td>hit</td>
</tr>
</tbody>
</table>

Victim’s Data

Attacker’s Data

Main Memory

Memory block 64 bytes

Taken from the presentation of “3D Integration: New Opportunities in Defense Against Cache-timing Side-channel Attacks” by Chongxi Bao and Ankur Srivastava on ICCD’15.
Limitations

- Can only be applied in small caches (L1 caches)

- Since it is used in small caches, it is applicable to processes located in the same core

4KB pages

Virtual Page

Offset

MMU

Physical Page

Cache tag

Set

Byte

Cache line size = 64 bytes
Offset for cache = 6 bits
Cache index = 6 bits
at most to access 64 sets
Practical Scenario

- In Cloud computing environment two users can share same hardware

- Users running on different cores share the last level cache
S$A attack (Shared Cache Attack)

- S$A attack is targeted towards the LLC
- Make use of huge size pages
- L1 – 64 sets
- L2 – 512 sets
- L3 – 4096 sets
- Takes advantage of the control of lower bits of the virtual address

Gorka Irazoqui, Thomas Eisenbarth and Berk Sunar, “S$A: A Shared Cache Attack that Works Across Cores and Defies VM Sandboxing—and its Application to AES”, Oakland’15
Steps involved in S$A attack

1. Allocation of huge size pages
   - Spy process have access to huge pages using his administrator rights in guest OS

2. Prime desired set in last level cache
   - Attacker creates data that fills a set in the LLC and primes it

3. Reprime
   - Since LLCs are inclusive some sets in the upper level will also be filled
   - Evict data from the upper level caches
   - Reprime data to fill different set in LLC but same set in upper level cache
Steps involved in S$A$ attack

4. Victim process runs
   • Victim runs the target process
   • If monitored cache set is used, some of the primed lines will be evicted
   • Else all primes lines will reside in the LLC

5. Probe and measure
   • After execution of victim process, spy process probes the primed memory lines and measures the time to probe
   • If one or more lines have been evicted probe time will be higher
   • Shorter probe time if no lines were evicted
Flush + Reload Attack

- Flush one cache line and time the execution of reloading the value to figure out whether the victim program has access this cache line or not.

- Fine-grained: attack at cache line granularity.

Yarom Y, Falkner K. Flush+ reload: a high resolution, low noise, L3 cache side-channel attack. USENIX Security 14
Flush + Flush Attack

- The same idea as Flush + Reload attack.
  - Problem: incur too much cache misses in reloading process, which may be used as a signature to detect cache side channel attack

- Flush + Flush attack exploits the execution time of Flush instruction to learn whether the Flush instruction hit the cache or not.

- Flush instruction can abort early in case of a cache miss. In case of a cache hit, it has to trigger eviction on all local caches, so it would take longer.

- Attack at cache line granularity, but less accuracy than Flush + Reload

- More stealthier, because incur fewer cache misses

Cache Storage Channel Attack

Exploit the uncachable property of some cache lines.
Any write to uncachable address will not modify the value in cache line.

Suppose attacker has a pair of alias VA_c and VA_nc, which map to the same physical address PA in cache.
Depending on some secret values, the victim may access PA.

This storage channel is less noisy than timing channels.

A1) write(VA_c, 1)
A2) write(VA_nc, 0)
A3) call victim
A4) D = read(VA_nc)
Cache interferences are the root causes of cache side channel attacks.

Software-based approaches are all attack specific and algorithm specific.

Hardware-based approaches:
- Randomize the cache interferences -> no information leakage through interference.
- Partition the cache statically -> no cache interferences.
RPcache (Random Permutation Cache)

- Randomizes cache-memory mapping, when a cache interference occurs, so no useful information about which cache line was evicted can be inferred.

Cache access handling procedure

- **Hit?**
  - Yes: Normal cache hit handling procedure
    - Update $P_D$
  - No: Choose $R$ for replacement, based on replacement policy
    - $R$ belongs to the current process?
      - Yes: $P_R = P_D$?
        - Yes: Normal cache miss handling procedure
          - perform the Id/st operation without replacing a cache line
        - No: Randomly select set $S'$; replace $R'$ in $S'$
      - No: Randomly select set $S'$; swap the mappings of $S$ and $S'$; Fix mappings for lines already in $S$ and $S'$

- **No**
  - Randomly select set $S'$; replace $R'$ in $S'$; swap the mappings of $S$ and $S'$; Fix mappings for lines already in $S$ and $S'$
  - Update $P_D$

**Figure 6. Cache access handling procedure for RPcache**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$, $S$</td>
<td>$R$ is the cache line being replaced in cache set $S$.</td>
</tr>
<tr>
<td>$R'$, $S'$</td>
<td>$R'$ is the cache line being replaced in another cache set $S'$ which is randomly selected.</td>
</tr>
<tr>
<td>$D$</td>
<td>The memory block being fetched into the cache.</td>
</tr>
<tr>
<td>$P_X$</td>
<td>The $P$-bit of cache line $X$, e.g., of $R$, $R'$ or $D$.</td>
</tr>
</tbody>
</table>
Figure 5. A logical view of the RPaCache

Table 3. Timing and Power Estimation of RPaCache

<table>
<thead>
<tr>
<th>RPaCache</th>
<th>16K 2way</th>
<th>32K 2way</th>
<th>16K 4way</th>
<th>32K 4way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access time (ns)</td>
<td>1.225 (+2.1%)</td>
<td>1.331 (+1.7%)</td>
<td>1.293 (+1.1%)</td>
<td>1.344 (+3.3%)</td>
</tr>
<tr>
<td>Power (nj)</td>
<td>1.205 (+8.6%)</td>
<td>1.282 (+1.3%)</td>
<td>1.792 (+6.1%)</td>
<td>1.906 (+2.1%)</td>
</tr>
</tbody>
</table>
Example of RPCache

1. Attacker fills set 1.
2. Attacker runs the encryption process.
3. Victim’s data maps to set 2 instead of set 1, and the mapping is swapped.
4. Attacker tries to access his data, and the mapping is swapped randomly again, so the hit rate of attacker’s data does not infer any memory access of victim.
PLCache (Partition-Locked Cache)

- A process is able to lock the cache lines in the cache, so the cache will not evicted by the data of other processes.

![Figure 3. A cache line of the PLcache](image)
Cache access handling procedure

Figure 4. Access handling procedure for PLcache
Performance Evaluation

**RPCache:** The performance impact caused by the random cache evictions in RPcache is negligible: worst case 1.7% (on 4K directed-mapped cache) and 0.3% on average.

**PLCache:** When the size of the protected memory (5KB) is larger than the cache capacity (4KB cache), the performance is always bad because all cache lines are locked. Set-associativity affects performance as well, direct-mapped cache has ~30% overhead.
Attack on PL Cache

- PL cache can protect the cache lines from evicting from the cache by other processes, but it does not prevent the cache access when we start loading the victim’s cache line.

- Evict + Time does not work any more.

- Prime + probe still works.

- Flush + Reload still works.

- Flush + Flush still works

Sanctum

- Sanctum offers strong provable isolation of software modules running concurrently and sharing resources, but protects against the attacks that infer private information from a program’s memory access patterns, including cache side channel attacks.

- Like SGX, Sanctum isolates the software inside an enclave from any other software on the system, including privileged system software.

Costan V, Lebedev I, Devadas S. Sanctum: Minimal Hardware Extensions for Strong Software Isolation[J].
Addresses in a DRAM region do not collide in the last level cache with addresses from any other DRAM region. So the OS can place two different applications in two different DRAM regions, then the cache interference in the last level cache is eliminated.

For high level caches, Sanctum flushes them whenever a core jumps between enclave and non-enclave code.
The fragmentation of DRAM regions makes it difficult for the OS to allocate contiguous DRAM buffers, which are essential to the efficient DMA transfers used by high performance devices. Shifting the physical page number by 3 bits yields contiguous DRAM regions.

Figure 5: Cache address shifting makes DRAM regions contiguous
**Sanctum**: Largest overhead is 4%, and average is 1.9% on an insecure baseline.

**Figure 16**: Sanctum’s enclave overheads for one core utilizing 1/4 of the LLC compared against an idealized baseline (non-enclave app using the entire LLC), and against a representative baseline (non-enclave app sharing the LLC with concurrent instances).
Language-based Approach

- Avoid timing channel during design phase
  - **SecVerilog**

```verilog
// Avoid timing channel during design phase

always @(posedge clock) begin
  if (write_enable) begin
    case (way)
      0: begin tag0[index]=tag_in; end
      1: begin tag1[index]=tag_in; end
      2: begin tag2[index]=tag_in; end
      3: begin tag3[index]=tag_in; end
    endcase
  end
end
```

(a) SecVerilog code for cache tags

```verilog
wire[L] isLoad,isStore;
wire[L] hit0,hit1; // hitX: 1 iff way X gets a cache hit
wire[H] hit2,hit3;
wire[L] hit=(timingLabel == 0) ? ((hit0||hit1)||hit2||hit3)?1:0;
assign stall = ((isLoad || isStore) & (~hit || (dFsmState != DFSM_IDLE)));
```

(b) SecVerilog code for a cache controller

Demo

- Prime + Probe attack on AES
- Key: 00 00 00 00 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

1. Introduction
   1. Timing channel
   2. Power channel
   3. EM radiation channel
   4. Acoustic channel
   5. Photonic emission channel

2. Timing attack on RSA
   1. Background
   2. Target RSA implementation
   3. Attack
   4. Defense
   5. Demo

3. Cache timing attack
   1. Attacks
   2. Countermeasures
   3. Demo

4. Differential Power Analysis
   1. Differential Power Analysis on AES
Complementary metal oxide semiconductor (CMOS) is a technology for constructing integrated circuits.

Since the static power consumption of CMOS is very low, CMOS processes have come to dominate. And now the vast majority of modern integrated circuit manufacturing is on CMOS processes.

BUT, this advantage can also be used by attackers.

The key idea is that the dynamic power consumption will be distinct from static power consumption.

So, high power consumption means a change from 0 -> 1 or 1 -> 0.
Power Model

- Higher power consumption
- = More bits flipping
- = Bigger Hamming Distance between input and output of the last round
Metric

- Correlation coefficient between real power consumption and Hamming Distance.

\[ r_{i,j} = \frac{\sum_{d=1}^{D} (h_{d,i} - \bar{h}_i) \cdot (t_{d,j} - \bar{t}_j)}{\sqrt{\sum_{d=1}^{D} (h_{d,i} - \bar{h}_i)^2 \cdot \sum_{d=1}^{D} (t_{d,j} - \bar{t}_j)^2}} \]
Workflow (continued)
Workflow (continued)
FPGA Board (SASEBO-G)
Experiment Equipment

- One DC power supply (Agilent E3610A)
- One digital oscilloscope (YOKOGAWA DL7200)
- One FPGA board (SASEBO-G)
- One PC
- Two probes
Experiment Setup

- 1. Configure the PC as an FTP server.
- 2. Build an Ethernet to connect digital oscilloscope and PC.
Experiment Setup (continued)

- 3. Download .bit files to FPGA board, one is used for AES operation and the other one is used to control the AES operation on the other chip.

- 4. Grab the trigger signal with the probe connected to channel 3. Take the power consumption waveform from a resistor paralleling with the AES chip via the channel 1.
5. Configure the digital oscilloscope. For channel 1, set the vertical scale to 50 mV/div, the offset to 150 mV, and enable 20MHz BWL. For channel 3, set the vertical scale to 1 V/div and the offset to 0 V. Set the trigger source to channel 3 and the triggering mode to negative edge.
Power Measurement

- Use a software called SASEBO-checker to launch AES operation and store the ciphertext which are feedback from FPGA.
Power Measurement (continued)

- Measure the power traces at a sampling rate of 2GHz, and store the power traces to PC via Ethernet.
- In total, we measured 10,000 power traces.
Data Analysis

- Write C code to compute the Hamming Distance and the correlation coefficient between real power consumption and hypothesis.
- Plot graphs of the correlation coefficients.
Results (byte 0) roundkey=13
Results (byte 1) roundkey=11
Results (byte 4) roundkey=E3
Results (byte 5) roundkey=94
Results (byte 8) roundkey=F3
Results (byte 9) roundkey=07
Results (byte 12) roundkey=4D
Results (byte 13)
roundkey=2B


5. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results". CHES’01


8. David Brumleya and Dan Boneh, “Remote timing attacks are practical". Computer Networks’05.


Reference

Reference


