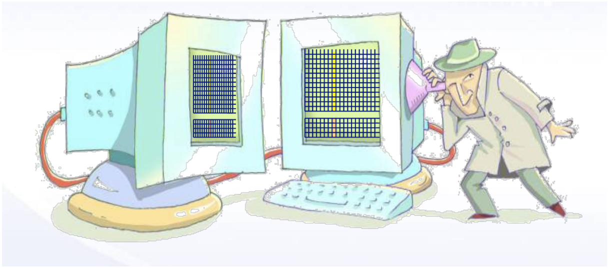
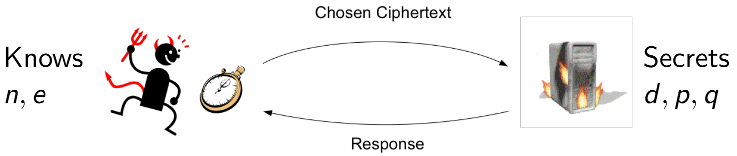
# Micro-Architectural Attacks and Countermeasures



## Contents

## Micro-Architectural Attacks

- Historical targets of side-channel attacks were smart cards
- The vulnerability of computer systems (e.g., remote servers) was not known until Brumley and Boneh Attack (2003)
- This remote timing attack on RSA (OpenSSL implementation on a web server) shows the practicality of such attacks
- The attack revealed 1024-bit RSA key of a server over a LAN
- Remote RSA attack was improved, now requiring $< 100,000$ queries (while the original attack needed 1.4 million queries)



## Micro-Architectural Attacks

- Side-channel attacks can be applied to the PC as well
- This is rather interesting since maturing Trusted Computing efforts promise a “trusted environment” with isolated execution for applications, etc.
- These new side-channel attacks are different from embedded platforms
- The PC platform environment is quite different from the embedded security platforms.
- Only pure “unprivileged” software-based attacks are really interesting

## Micro-Architectural Attacks

- Since the power is not easily observable in a complicated device such as PC, the timing variations are targeted
- Timing variances exploited in timing attacks are caused by:
    - Different data-dependent execution paths
    - Different (number of) instructions executed in those paths
- In general, micro-architectural attacks exploit timing and access variations caused by the components of the CPU (even if the same sequence instructions are always executed)

## Micro-Architectural Attacks

- Micro-architectural side-channel attacks are a new class of attacks that exploit the micro-architecture and throughput-oriented internal functionality of modern processor components
- Micro-architectural attacks exploit the execution time variations caused by CPU components
- Currently there are 4 types of Micro-Architectural Attacks:
  - Cache Analysis
  - Branch Prediction Analysis
  - Instruction Cache
  - Shared Functional Units

## Micro-Architectural Attacks

- These attacks capitalize on the situations where several applications share the same processor resources
- The shared usage between spy and crypto process allows a spy process running in parallel to the victim process to extract critical information like secret keys.
- On powerful PC-platforms many applications can run in parallel:
  - Either quasi-parallel enabled by OS scheduling, or
  - More or less explicitly parallel depending on the degree of additional hardware, such dual processors, multicores, and simultaneous multi threading

## Micro-Architectural Attacks

- Thus, several applications share the same processor and its resources, and also at more or less the same time
- Therefore, when a highly critical crypto algorithm is executed, there is the potential threat that a malicious or so called spy process is executed in parallel with the crypto process which might try to extract critical or secret information by "spying" on the crypto process during its execution

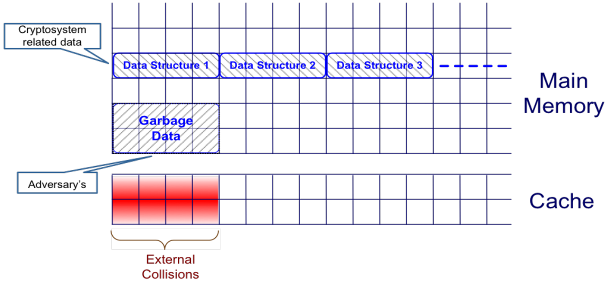
## Cache Attacks

- Cache is a small and fast storage area used by the CPU to reduce the average time to access main memory.
- It stores copies of the most frequently used data
- When a CPU needs to read a location in main memory, it first checks to see if the data is already in the cache
  - Cache Hit: data is already in the cache; CPU immediately uses this data in cache.
  - Cache Miss: data is not in the cache; CPU reads it from the memory and stores a copy in the cache.
- Cache Block: The minimum amount of data that can be read from the main memory into the cache at once
- Each cache miss causes a cache block to be retrieved from a higher level memory.

## Cache Attacks

- Cache attacks exploit the cache behavior (i.e., cache hit/miss statistics) of cryptosystems
- Cache architecture leaks information about memory access patterns
- The sources of information leakage:
  - Execution Time: cache misses take more time than a cache hit
  - Power Consumption: cache misses require more power than a cache hit
- Cryptosystems have data dependent memory access patterns
- Once the access patterns are extracted, an adversary may recover the secret key

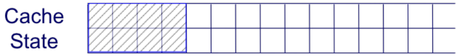
## Cache Attacks



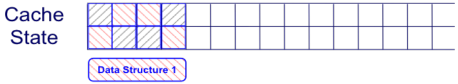
- An access to "Data Structure 1" may evict adversary's data and vice versa
- An adversary can infer if/when "Data Structure 1" is accessed during the encryption

## Cache Attacks

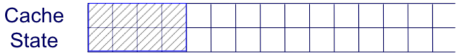
- Adversary reads the garbage data via the "Spy Process"



- Case 1: "Data Structure 1" is accessed



- Case 2: "Data Structure 1" is not accessed



- Spy process reads the garbage data again
  - Case 1: takes more time to read it
  - Case 2: takes less time to read it

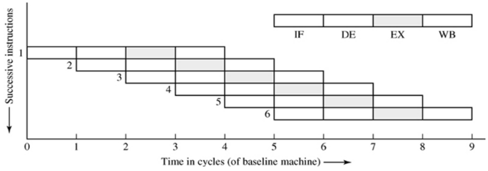
## Branch Prediction Attack (BPA)

- A very new software side-channel enabled by the branch prediction capability common to all modern CPUs
- The penalty paid (extra clock cycles) for a mispredicted branch can be used for cryptanalysis of cryptographic primitives that employ a data-dependent program flow
- BPA allows an unprivileged process to attack other processes running in parallel on the same processor
- Works despite of sophisticated partitioning methods such as memory protection, sandboxing or even virtualization
- Public-key ciphers like RSA and ECC are susceptible to BP attacks

## Branch Prediction Attack

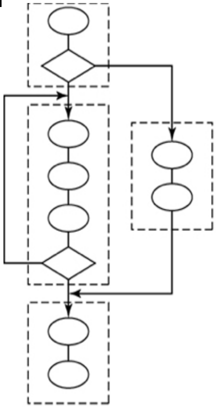
- Superscalar processors have to execute instructions speculatively to overcome control hazards
- Branch prediction units try to predict the most likely execution path after a branch
- A branch instruction is a point in the instruction stream of a program where the next instruction is not necessarily the next sequential one
- For conditional branches, the decision to take the branch or not to take depends on some condition that must be evaluated in order to make the correct decision
- During this evaluation period, the processor speculatively executes instructions from one of the possible execution paths instead of stalling and awaiting for the decision to come through

## Branch Prediction Attacks

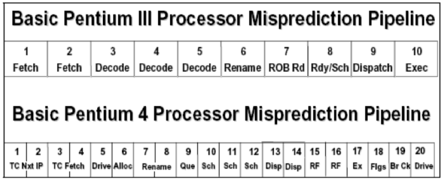


Due to the deep pipelining of the instruction sequences

A branch operation could stall the pipeline



Introducing misprediction delays up to 150 cycles



**Basic Pentium III Processor Misprediction Pipeline**

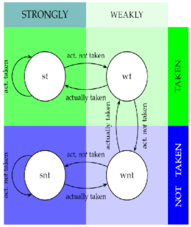
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Fetch | Fetch | Decode | Decode | Decode | Rename | ROB Rd | Rdy/Sch | Dispatch | Exec |

**Basic Pentium 4 Processor Misprediction Pipeline**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TC Nxt IP | | TC Fetch | | Drive | Alloc | Rename | | Que | Sch | Sch | Sch | Disp | Disp | RF | RF | Ex | Flgs | Br Ck | Drive |

## Branch Prediction Attack

- A branch predictor determines whether a conditional branch in the instruction flow of a program is likely to be taken or not



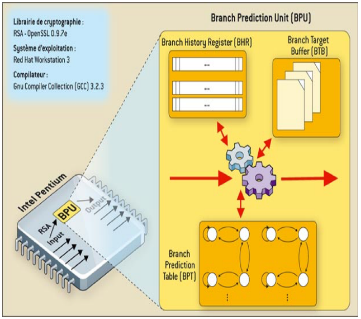
- Branch predictors are crucial in today's modern, superscalar processors for achieving high performance
- They allow processors to fetch and execute instructions without waiting for a branch to be resolved

- Almost all pipelined processors do branch prediction of some form, because they must guess the address of the next instruction to fetch before the current instruction has been executed
- Branch prediction is not the same as branch target prediction
- Branch target prediction attempts to guess the target of the branch or unconditional jump before it is computed by parsing the instruction itself

## Branch Prediction Attack

- BPU consists of mainly two “logical” parts: the branch target buffer (BTB) and the branch predictor logic

- BTB is the buffer where the CPU stores the target addresses of the previously executed branches
- BTB is limited in size, the CPU can store only a number of such target addresses, and previously stored addresses are evicted from the BTB if a new address needs to be stored instead
- The predictor is that part of the BPU that makes the prediction on the outcome of the branch


Librairie de cryptographie :
RSA - OpenSSL 0.9.7e
Système d'exploitation :
Red Hat Workstation 3
Compilateur :
Gnu Compiler Collection (GCC) 3.2.3
Branch Prediction Unit (BPU)
Branch History Register (BHR)
Branch Target Buffer (BTB)
Branch Prediction Table (BPT)
Intel Pentium
BPU
RSA
Input
Output

## Branch Prediction Attack

- BPA uses the branch misprediction delays to break cryptographic primitives using a data-dependent program flow
- BPA also allows an unprivileged process to attack other processes running in parallel on the same processor even in the presence of sophisticated partitioning methods such as memory protection, sandboxing or even virtualization
- 4 different branch prediction attacks are proposed:
  - Exploiting the Predictor directly (Direct Timing Attack)
  - Forcing the BPU to the Same Prediction (Asynchronous Attack)
  - Forcing the BPU to the Same Prediction (Synchronous Attack)
  - Trace-driven Attack against the BTB (Simple Prediction Attack)

## Direct Timing Attack (DTA)

- DTA relies on the fact that the prediction algorithms are deterministic (i.e., predictable)
- DTA assumes that an adversary attacks an RSA cipher with a private exponent *d* and knows the first (the most significant) *i* bits of *d* and is trying to reveal $d_i$.

```
$S = A * B$
$S = (S - (S * N^{-1} \bmod R) * N)/R$
**if** $S > N$ **then** $S = S - N$
**return** $S$
```

## Direct Timing Attack

- For any message $m$, the adversary can simulate the first $i$ steps of the operation and obtain the intermediate result that will be the input of the $(i+1)$th squaring
- Then, the attacker creates 4 different message sets $M_1$, $M_2$, $M_3$, and $M_4$, such that

$$\begin{aligned}
M_1 &= \{m \mid m \text{ causes a misprediction during MM of } (i+1)^{\text{th}} \text{ squaring if } d_i = 1\} \\
M_2 &= \{m \mid m \text{ does not cause a misprediction during MM of } (i+1)^{\text{th}} \text{ squaring if } d_i = 1\} \\
M_3 &= \{m \mid m \text{ causes a misprediction during MM of } (i+1)^{\text{th}} \text{ squaring if } d_i = 0\} \\
M_4 &= \{m \mid m \text{ does not cause a misprediction during MM of } (i+1)^{\text{th}} \text{ squaring if } d_i = 0\}
\end{aligned}$$

- If the difference between the average execution time of $M_1$ and $M_2$ is more significant than that of $M_3$ and $M_4$, then the attacker guesses that $d_i = 1$. Otherwise $d_i$ should be 0

## Implications of Micro-Architectural Attacks (MAA)

- The micro-architectural attacks work despite of sophisticated partitioning and protection methods such as memory protection, sandboxing, and virtualization
- They can impact: Multiuser systems, VPNs, Virtual machines, Trusted computing, Sandboxes (JVM, JavaScript), and Remote attacks
- They are:
  - Easy to deploy – pure software attacks
  - Hard to detect
  - Hard to protect efficiently

## Software Countermeasures against MAA

- OpenSSL had gone into several revisions and implemented many software countermeasures
- Yet, new micro-architectural vulnerabilities of OpenSSL are being discovered
- Are software countermeasures sufficient?
    - Solving the problem in software means solving it one by one
    - Software solutions are algorithm- and attack-specific
    - They incur high performance overhead

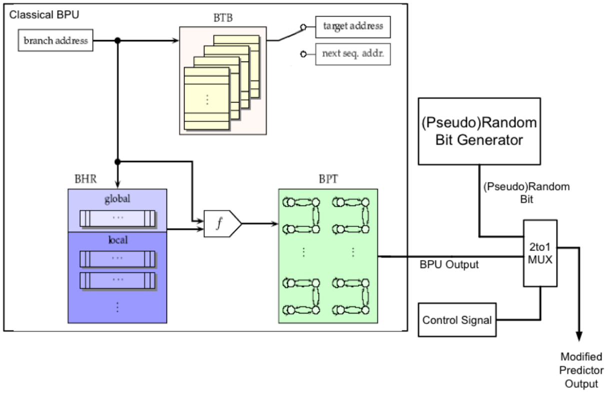
## Hardware Countermeasures against MAA

- Hardware countermeasures:
  - Solving it in hardware may mean solving it for all
  - They may not be algorithm-specific
  - They may have much less overhead
- Possible branch prediction countermeasures:
  - Randomizable branch prediction
  - Partitioned Branch Target Buffers
  - Locking mechanism for BTB
  - Protected BTB area
  - Flushing mechanism for BTB
  - Dynamically disabling branch predictions

## Hardware Countermeasures against MAA

Randomizable Prediction:

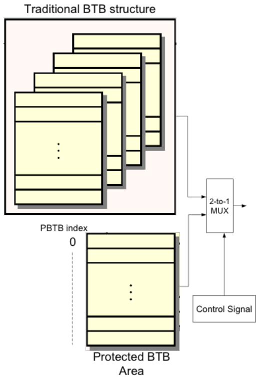
- Modify the prediction output
- BPU can output random predictions for critical branches



## Hardware Countermeasures against MAA

Protected Branch Target Buffer:

- Allow critical code to benefit from using a protected buffer
- The entries in PBTB can be handled in a more secure way


Traditional BTB structure
PBTB index
0
2-to-1 MUX
Control Signal
Protected BTB Area