Who watches the watchmen?: Utilizing Performance Monitors for Compromising keys of RSA on Intel Platforms

Sarani Bhattacharya and Debdeep Mukhopadhyay

Indian Institute of Technology Kharagpur

CHES 2015
September 15, 2015

Overview of the talk

- Introduction
- Motivation of the problem
- Exponentiation primitives for Public key cryptography
- Modelling branch misses as side-channel
- Formally modeling success probability
- Experimental validation
- Conclusion

Introduction

- Hardware performance counters (HPCs) are a set of special-purpose registers to store the counts of hardware-related activities within the microprocessor.

Introduction

- Hardware performance counters (HPCs) are a set of special-purpose registers to store the counts of hardware-related activities within the microprocessor.
- Hence HPCs can be utilized for both attacks and their countermeasures.

Introduction

- Hardware performance counters (HPCs) are a set of special-purpose registers to store the counts of hardware-related activities within the microprocessor.
- Hence HPCs can be utilized for both attacks and their countermeasures.
- Asymmetric-key cryptographic algorithms when implemented on systems with branch predictors, are subjected to side-channel attacks exploiting the deterministic branch predictor behaviour due to their key-dependent input sequences.

Objective of the work

- This work shows that HPCs, which are used as performance monitors (watchmen) in modern computer systems can be utilized to retrieve the secret keys by reasonably modelled adversaries.

Objective of the work

- This work shows that HPCs, which are used as performance monitors (watchmen) in modern computer systems can be utilized to retrieve the secret keys by reasonably modelled adversaries.
- The attack exploits the characteristics of branch predictor and shows formally that the leakage of the key increases with the ability of the attacker to model the predictor more accurately.

Objective of the work

- This work shows that HPCs, which are used as performance monitors (watchmen) in modern computer systems can be utilized to retrieve the secret keys by reasonably modelled adversaries.
- The attack exploits the characteristics of branch predictor and shows formally that the leakage of the key increases with the ability of the attacker to model the predictor more accurately.
- We claim that branch misses from HPCs are indeed more significant side-channels compared to timing.

Why should we consider HPCs for security analysis?

- Results from HPCs are treated as an accurate representations of events occurring in hardware [1], [2].

Why should we consider HPCs for security analysis?

- Results from HPCs are treated as an accurate representations of events occurring in hardware [1], [2].
- This occurs when the overhead introduced by performance counter interfaces does not dominate the event counts.

Why should we consider HPCs for security analysis?

- Results from HPCs are treated as an accurate representations of events occurring in hardware [1], [2].
- This occurs when the overhead introduced by performance counter interfaces does not dominate the event counts.
- The accuracy depends upon the interface used, the application and the event being measured [1].

Exploiting Hardware Performance Counters

- HPC L1 and L2 D-cache miss counters have been exploited as side-channels in [3] for performing timing based cache attacks on symmetric-key algorithms, like AES.

Exploiting Hardware Performance Counters

- HPC L1 and L2 D-cache miss counters have been exploited as side-channels in [3] for performing timing based cache attacks on symmetric-key algorithms, like AES.
- On the other hand, in [4] data from performance counters are used to develop a malware detector in hardware using machine learning techniques.

Exploiting Hardware Performance Counters

- HPC L1 and L2 D-cache miss counters have been exploited as side-channels in [3] for performing timing based cache attacks on symmetric-key algorithms, like AES.
- On the other hand, in [4] data from performance counters are used to develop a malware detector in hardware using machine learning techniques.
- While in [5], a new Virtual Machine Monitor (VMM) named NumChecker is proposed, which exploits HPCs to detect kernel root-kits in a guest Virtual Machine.

Performance Monitoring over the years

In [6], profiling HPCs are referred to be accessible in high-privilege modes. But since the advent of Linux-Perf for userspace Program Analysis [7], [8] this highly accurate performance monitoring information are available to Linux users from supercomputers to embedded systems.

Performance Monitoring over the years

In [6], profiling HPCs are referred to be accessible in high-privilege modes. But since the advent of Linux-Perf for userspace Program Analysis [7], [8] this highly accurate performance monitoring information are available to Linux users from supercomputers to embedded systems.

- Oprofile- a system-wide sampling profiler by Levon which was included into Linux 2.5.43 in 2002.

Performance Monitoring over the years

In [6], profiling HPCs are referred to be accessible in high-privilege modes. But since the advent of Linux-Perf for userspace Program Analysis [7], [8] this highly accurate performance monitoring information are available to Linux users from supercomputers to embedded systems.

- Oprofile- a system-wide sampling profiler by Levon which was included into Linux 2.5.43 in 2002.
- PAPI implementation for Linux uses the perfctr Linux patch an event-monitoring device driver to enable access to the counters.

Performance Monitoring over the years

In [6], profiling HPCs are referred to be accessible in high-privilege modes. But since the advent of Linux-Perf for userspace Program Analysis [7], [8] this highly accurate performance monitoring information are available to Linux users from supercomputers to embedded systems.

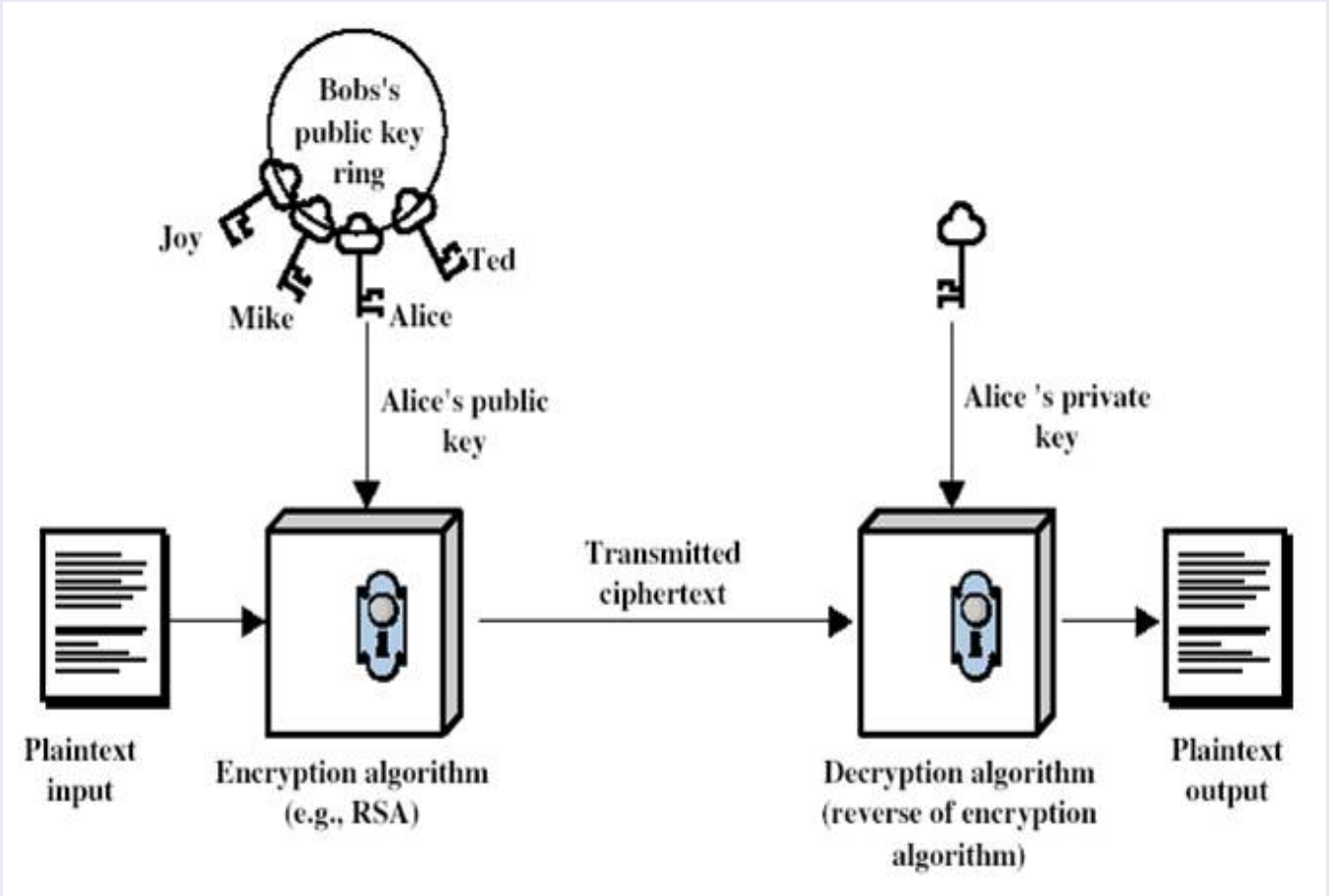
- Oprofile- a system-wide sampling profiler by Levon which was included into Linux 2.5.43 in 2002.
- PAPI implementation for Linux uses the perfctr Linux patch an event-monitoring device driver to enable access to the counters.
- In 2009, event named 'perf' subsystem was added to the Linux kernel, and makes user access to performance counters less clumsy, without kernel patches or recompiles [9].

Performance Monitoring over the years

In [6], profiling HPCs are referred to be accessible in high-privilege modes. But since the advent of Linux-Perf for userspace Program Analysis [7], [8] this highly accurate performance monitoring information are available to Linux users from supercomputers to embedded systems.

- Oprofile- a system-wide sampling profiler by Levon which was included into Linux 2.5.43 in 2002.
- PAPI implementation for Linux uses the perfctr Linux patch an event-monitoring device driver to enable access to the counters.
- In 2009, event named 'perf' subsystem was added to the Linux kernel, and makes user access to performance counters less clumsy, without kernel patches or recompiles [9].
- Greatest advantage of Perf event [9] is the subsystem has been already included in the Linux kernel 2.6.31 as "Performance Counters for Linux".

Public key Cryptography



Exponentiation and Underlying Multiplication Primitive

- Inputs(M) are encrypted and decrypted by performing modular exponentiation with modulus N on public or private keys represented as n bit binary string.

Square and Multiply Exponentiation

Algorithm 1: **Binary version of Square and Multiply Exponentiation Algorithm**

$S \leftarrow M$;
for i **from** 1 **to** $n-1$ **do**
    $S \leftarrow S * S \ mod \ N$;
    if $d_i = 1$ **then**
        $S \leftarrow S * M \ mod \ N$;
    end
end
return S ;

- Conditional execution of instruction and their dependence on secret exponent is exploited by the simple power and timing side-channels [10].

Montgomery Ladder Exponentiation Algorithm

- A naïve modification is to have a balanced ladder structure having equal number of squarings and multiplications.
- Most popular exponentiation primitive for Asymmetric-key cryptographic implementations.

Algorithm 2: Montgomery Ladder Algorithm

$R_0 \leftarrow 1$;
$R_1 \leftarrow M$;
for i **from** 0 **to** $n-1$ **do**
  if $d_i = 0$ **then**
    $R_1 \leftarrow (R_0 * R_1) \mod N$;
    $R_0 \leftarrow (R_0 * R_0) \mod N$;
  end
  else
    $R_0 \leftarrow (R_0 * R_1) \mod N$;
    $R_1 \leftarrow (R_1 * R_1) \mod N$;
  end
end
return R_0 ;

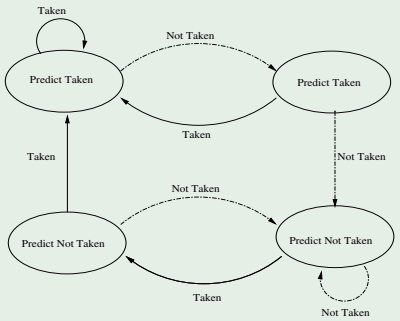
Montgomery Multiplication Algorithm

- Highly efficient algorithm for performing modular squaring and modular multiplication operation [11].
- Avoids time consuming integer division operation.
- R is assumed to be 2^k, when N is k-bit number.
- Calculates $Z = A * B * R^{-1} (\mathrm{mod} N)$, $A = a * R (\mathrm{mod} N)$, $B = b * R (\mathrm{mod} N)$ and $R^{-1} * R = 1 (\mathrm{mod} N)$.

Algorithm 3: Montgomery Multiplication Algorithm

$S \leftarrow A * B$;
$S \leftarrow (S + (S * N^{-1} \ mod \ R) * N)/R$;
if $S > N$ **then**
    $S \leftarrow S - N$;
end
return S ;

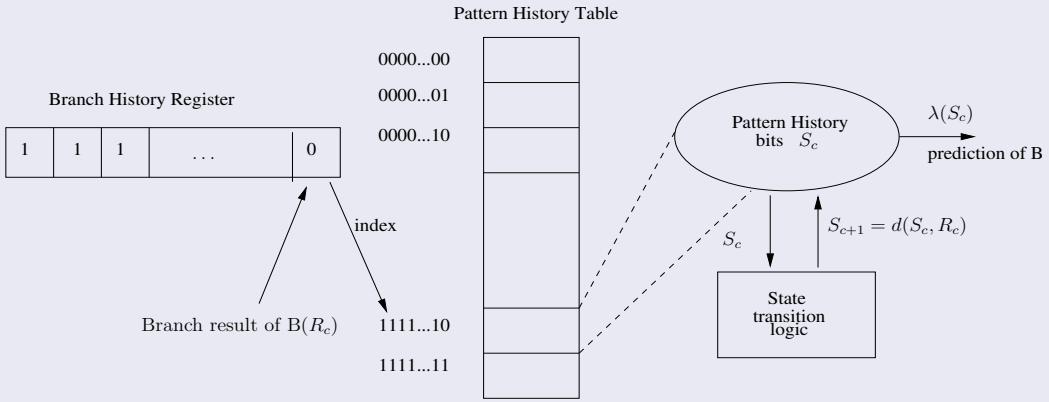
Branch Predictor State Machines



Dynamic 2-bit predictor State Machine

- The predictor must miss twice before the prediction changes.
- Conditional branching in regular recurring fashion goes undetected.

Two Level Adaptive Branch Prediction [12]



Modelling Branch Miss as Side-Channel from HPC

- We monitor the branch misses on the square and multiply and Montgomery Ladder algorithm using Montgomery multiplication as subroutine for operations like squaring and multiplication.
- Branch miss rely on the ability of branch predictor to correctly predict future branches to be taken.
- Profiling of HPCs using performance monitoring tools provides simple user interface to different hardware event counts and are considered as side-channel.

Approximating the System predictor with 2-bit branch predictor

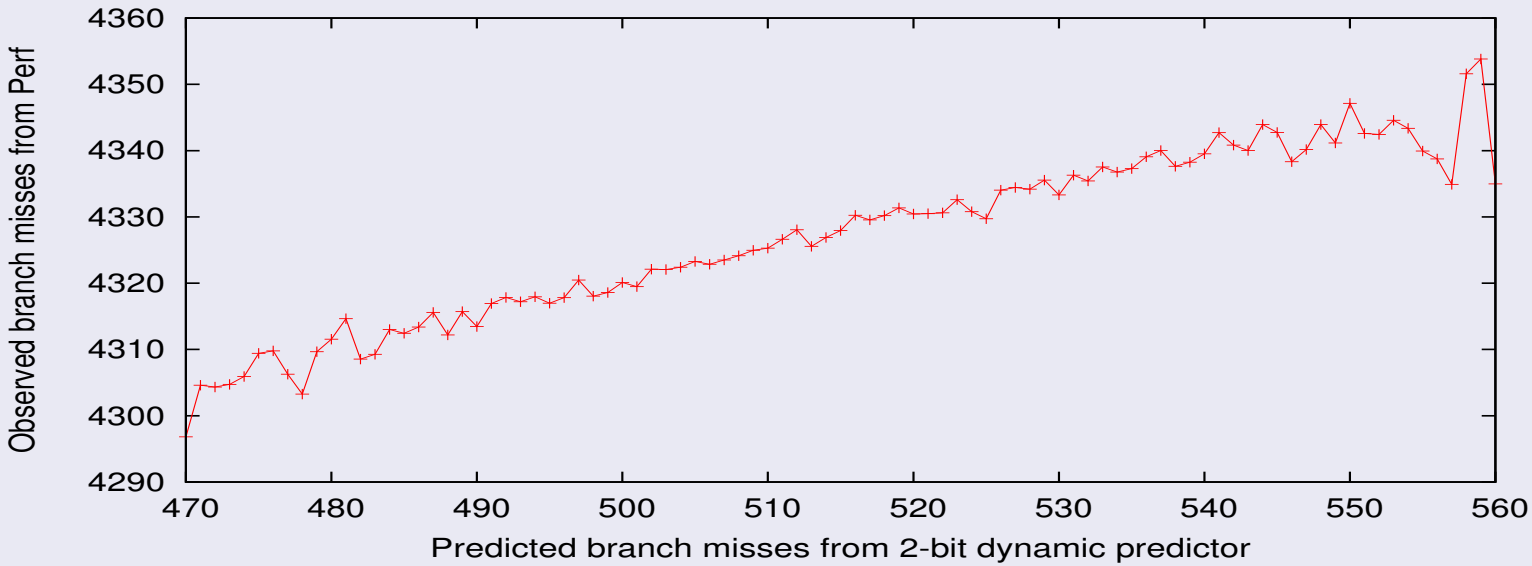


Figure: Variation of branch-misses from performance counters with increase in branch miss from 2-bit predictor algorithm

- Direct correlation observed for the branch misses from HPCs and from the simulated 2-bit dynamic predictor over a sample of exponent bitstream.
- This confirms assumption of 2-bit dynamic predictor being an approximation to the underlying system branch predictor.

Idea of the Attack

- In [13], timing attack exploiting branch mispredictions are demonstrated which requires the knowledge of actual structure of branch prediction hardware of the target system.
- Advantage of this attack lies in the fact that adversary, inspite of having no knowledge of the underlying architecture, can actually target real systems and reveal secret exponent bits, exploiting the branch miss as side-channel from HPCs.
- This is an iterative attack, targeting i^{th} bit assuming previous bits to be known.
- The attack separates a sample input set based on mispredictions for conditional reduction of Montgomery multiplication at the $(i+1)^{th}$ squaring step of exponentiation assuming secret i^{th} bit.

Threat Model for the attack

- The attacker knows first i bits of the private key and wants to determine next unknown bit d_i of the key $(d_0, d_1, \cdots, d_i, \cdots, d_{n-1})$.
- Generate a trace of branches as $(t_{m,1}, t_{m,2}, \cdots, t_{m,i})$ for conditional reduction of Montgomery multiplication at every squaring step.
- Under the assumption of d_i having value j, where $j \in \{0, 1\}$, appropriate value of $t^j_{m,i+1}$ is simulated.

Offline Phase of the Attack

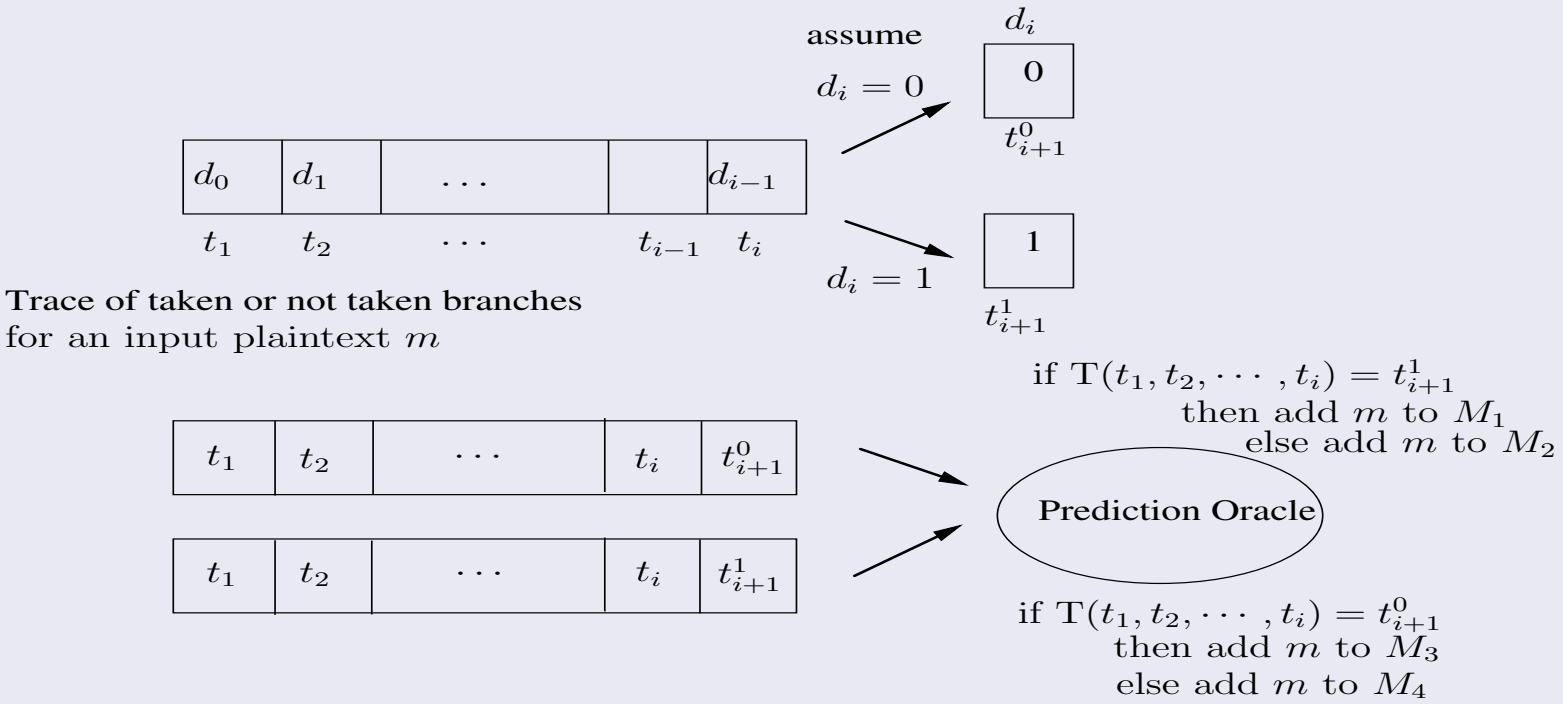


Figure: Partitioning randomly generated Ciphertexts set based on simulated Branch miss Modelling

Separation of Random Inputs

1. $M_1 = \{m | m$ does not cause a miss during MM of $(i+1)^{th}$ squaring if $d_i = 1\}$
2. $M_2 = \{m | m$ causes a misprediction during MM of $(i+1)^{th}$ squaring if $d_i = 1\}$
3. $M_3 = \{m | m$ does not cause a miss during MM of $(i+1)^{th}$ squaring if $d_i = 0\}$
4. $M_4 = \{m | m$ causes a misprediction during MM of $(i+1)^{th}$ squaring if $d_i = 0\}$

We ensure that there must be no common ciphertexts in sets (M_1, M_3) and (M_2, M_4) and the sets should be disjoint.

Online Phase

The probable next bit is decided following the Algorithm 4.

- If$(avg(\mathcal{M}_{M_2}) > avg(\mathcal{M}_{M_1}))$ and $(avg(\mathcal{M}_{M_4}) < avg(\mathcal{M}_{M_3}))$, then the next bit $(nb_i) = 1$
- Otherwise, if $(avg(\mathcal{M}_{M_4}) > avg(\mathcal{M}_{M_3}))$ and $(avg(\mathcal{M}_{M_2}) < avg(\mathcal{M}_{M_1}))$ then, next bit $(nb_i) = 0$

Algorithm 4: Adversary Attack Algorithm

Input: $(d_0, d_1, \cdots, d_{i-1}), M$
Output: Probable next bit nb_i
begin
  Offline Phase;
  for $\forall m \in M$ **do**
    Generate taken/ not-taken trace for input m as $t_{m,1}, t_{m,2}, \cdots, t_{m,i}$;
    Assume $d_i = 0$ and 1, generate $t^0_{m,i+1}$, $t^1_{m,i+1}$ respectively;
    $p_{m,i+1} = T(t_{m,1}, t_{m,2}, \cdots, t_{m,i})$;
    if $p_{m,i+1} = t^1_{m,i+1}$ **then**
      Add m to M_1 ;
    end
    else
      Add m to M_2 ;
    end
    if $p_{m,i+1} = t^0_{m,i+1}$ **then**
      Add m to M_3 ;
    end
    else
      Add m to M_4 ;
    end
  end
  Remove Duplicate Ciphertexts in the sets M_1, M_3 and M_2, M_4;
  Online Phase;
  Observe distribution of branch misses from performance counters as $\mathcal{M}_{M_1}, \mathcal{M}_{M_2}, \mathcal{M}_{M_3}, \mathcal{M}_{M_4}$;
  if $(avg(\mathcal{M}_{M_2}) > avg(\mathcal{M}_{M_1}))$ *and* $(avg(\mathcal{M}_{M_4}) < avg(\mathcal{M}_{M_3}))$ **then**
    $nb_i = 1$;
  end
  if $(avg(\mathcal{M}_{M_4}) > avg(\mathcal{M}_{M_3}))$ *and* $(avg(\mathcal{M}_{M_2}) < avg(\mathcal{M}_{M_1}))$ **then**
    $nb_i = 0$;
  end
  return nb_i ;
end

Formally Modelling the Success Probability

In the offline phase

- Assuming $d_i = 1$
  $\Pr[m_1 \in M_1] = \Pr[p_{m_1,i+1} = t^1_{m_1,i+1}]$
  $\Pr[m_2 \in M_2] = \Pr[p_{m_2,i+1} \neq t^1_{m_2,i+1}]$
- Assuming $d_i = 0$
  $\Pr[m_3 \in M_3] = \Pr[p_{m_3,i+1} = t^0_{m_3,i+1}]$
  $\Pr[m_4 \in M_4] = \Pr[p_{m_4,i+1} \neq t^0_{m_4,i+1}]$
- After removing duplicates, $t^0_{m,i+1} \neq t^1_{m,i+1}$.

In the online phase

Let nb_i be the bit which the attacker concludes to be the next secret bit. Let the expectation of the distribution of branch misses $(\mathcal{M}_M, \forall m \in M)$ be $\overline{\mathcal{M}_M}$. Thus,

$\Pr[nb_i = 0] = \Pr[(\overline{\mathcal{M}_{M_4}} - \overline{\mathcal{M}_{M_3}}) > 0 \wedge (\overline{\mathcal{M}_{M_2}} - \overline{\mathcal{M}_{M_1}}) < 0]$

$\Pr[nb_i = 1] = \Pr[(\overline{\mathcal{M}_{M_2}} - \overline{\mathcal{M}_{M_1}}) > 0 \wedge (\overline{\mathcal{M}_{M_4}} - \overline{\mathcal{M}_{M_3}}) < 0]$.

Formally Modelling the Success Probability

- Let $(i+1)^{th}$ branch predicted by the real predictor for input m is $r_{m,i+1}$.
- Let $i+1^{th}$ branch instruction has trace $B_{m,i+1}$ for unknown bit d_i.
- If $d_i = 0$, then $B_{m,i+1} = t^0_{m,i+1}$, otherwise if $d_i = 1$, $B_{m,i+1} = t^1_{m,i+1}$.
- Thus we can rewrite the previous equation as

$$\Pr[nb_i = 0] = \Pr[(\overline{\mathcal{M}_{M_4}} - \overline{\mathcal{M}_{M_3}}) > 0 \wedge (\overline{\mathcal{M}_{M_2}} - \overline{\mathcal{M}_{M_1}}) < 0]$$
$$= \Pr[(r_{m_4,i+1} \neq B_{m_4,i+1}) \wedge (r_{m_3,i+1} = B_{m_3,i+1}) \wedge (r_{m_2,i+1} = B_{m_2,i+1}) \wedge (r_{m_1,i+1} \neq B_{m_1,i+1})]$$

Formally Modelling the Success Probability

$$\begin{aligned}\Pr(\text{Success}) &= \Pr[nb_i = d_i] \\ &= \Pr[nb_i = 0 \wedge d_i = 0] + \Pr[nb_i = 1 \wedge d_i = 1] \\ &= \Pr[nb_i = 0 \mid d_i = 0] \cdot \Pr[d_i = 0] + \Pr[nb_i = 1 \mid d_i = 1] \cdot \Pr[d_i = 1]\end{aligned}$$

If $d_i = 0$, we replace $B_{m,i+1} = t^0_{m,i+1}$ in Equation 1 as,

$$\begin{aligned}\Pr[nb_i = 0 \mid d_i = 0] &= \Pr[(r_{m_4,i+1} \neq t^0_{m_4,i+1}) \wedge (r_{m_3,i+1} = t^0_{m_3,i+1}) \wedge (r_{m_2,i+1} = t^0_{m_2,i+1}) \wedge (r_{m_1,i+1} \neq t^0_{m_1,i+1})] \\ &= \Pr[(r_{m_4,i+1} \neq t^0_{m_4,i+1}) \wedge (r_{m_3,i+1} = t^0_{m_3,i+1}) \wedge (r_{m_2,i+1} \neq t^1_{m_2,i+1}) \wedge (r_{m_1,i+1} = t^1_{m_1,i+1})] \\ &\qquad \text{(since } t^0_{m_2,i+1} \neq t^1_{m_2,i+1} \text{ and } t^1_{m_1,i+1} \neq t^0_{m_1,i+1}\text{)}\end{aligned}$$

Substituting the events from Offline Phase,

$$\begin{aligned}\Pr[nb_i = 0 \mid d_i = 0] &= \Pr[(r_{m_4,i+1} = p_{m_4,i+1}) \wedge (r_{m_3,i+1} = p_{m_3,i+1}) \wedge (r_{m_2,i+1} = p_{m_2,i+1}) \wedge (r_{m_1,i+1} = p_{m_1,i+1})] \\ &= \Pr[(r_{m,i+1} = p_{m,i+1})]\end{aligned}$$

Formally Modelling the Success Probability

Similar calculations reveal,

$$\Pr[nb_i = 1 \mid d_i = 1] = \Pr[(r_{m,i+1} = p_{m,i+1})]$$

Combining equations we get,

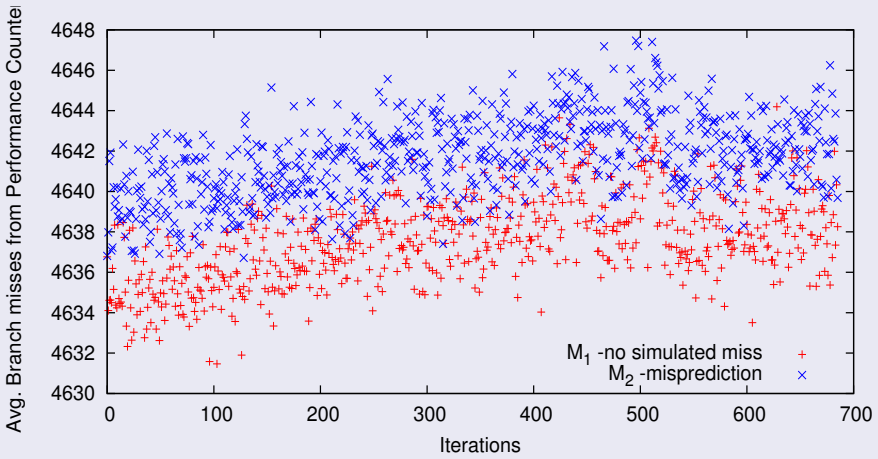
$$\begin{aligned} \Pr(\text{Success}) &= \Pr[r_{m,i+1} = p_{m,i+1}] \cdot [\Pr(d_i = 0) + \Pr(d_i = 1)] \\ &= \Pr[r_{m,i+1} = p_{m,i+1}] \end{aligned}$$

Thus the probability of success is equal to the probability that the theoretical predictor closely models the real predictor.

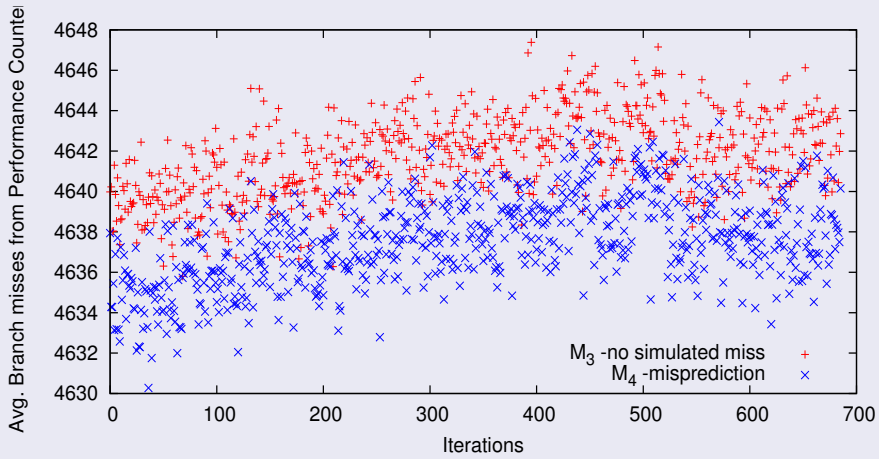
Experimental Validation for the Online Phase of the Attack

- A large input set is separated by simulations over bimodal and two-level adaptive predictor.
- Average branch misses are observed from HPCs for each elements in set M_1 , M_2 , M_3 and M_4.
- Each set has $L = 1000$ elements.
- Experiment is repeated over $I = 1000$ iterations.
- Experiments are performed on various platforms as Core-2 Duo E7400, Intel Core i3 M350 and Intel Core i5-3470.

Experiments on Square and Multiply Algorithm



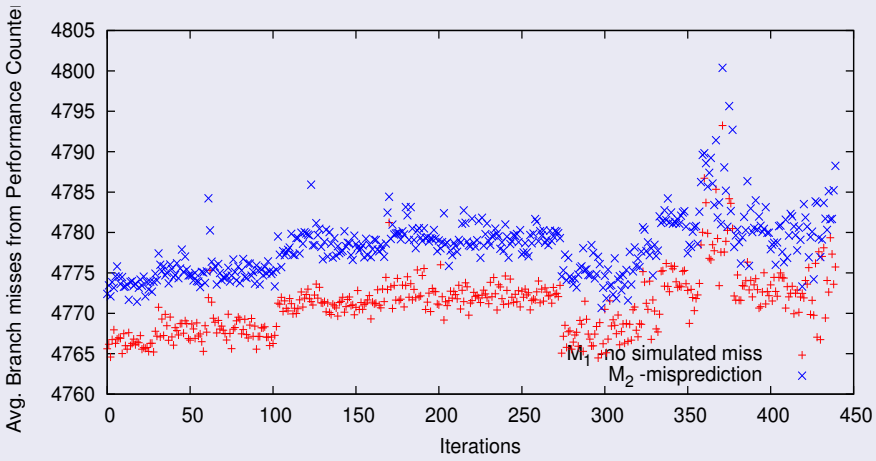
(a) **Correct Assumption** $d_i = 1$



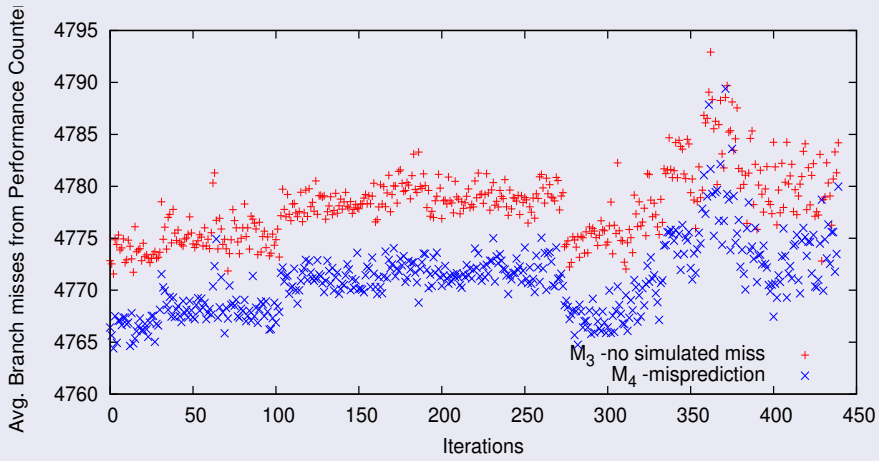
(b) **Incorrect Assumption** $d_i = 0$

Figure: Branch misses from HPCs on square and multiply correctly identifies secret bit $d_i = 1$, ciphertext set partitioned by simulated misses of two-level adaptive predictor

Experiments on Montgomery Ladder



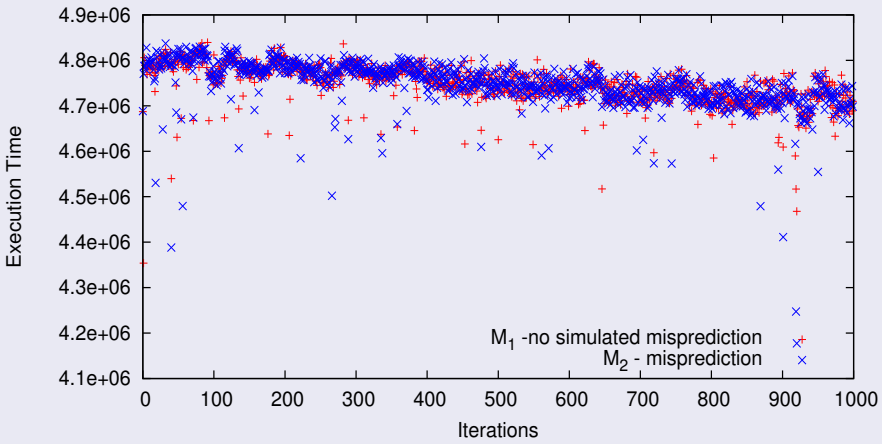
(a) **Correct Assumption** $d_i = 1$



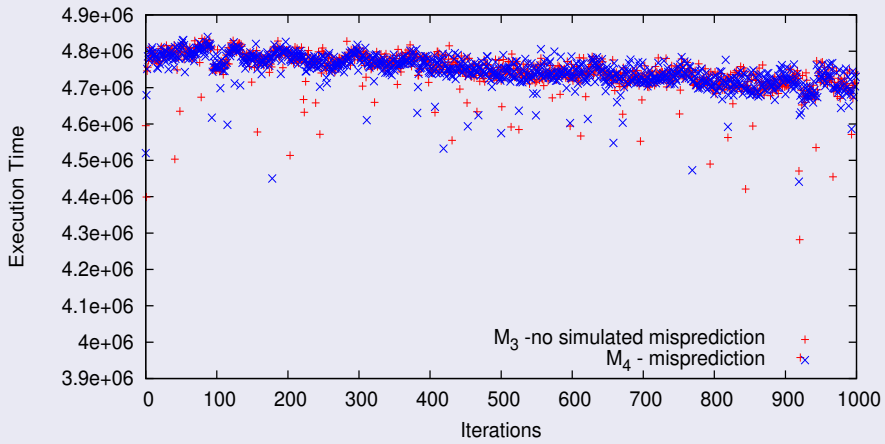
(b) **Incorrect Assumption** $d_i = 0$

Figure: Branch misses from HPCs on Montgomery Ladder correctly identifies secret bit $d_i = 1$, ciphertext set partitioned by simulated misses of two-level adaptive predictor

Comparison with timing as side-channel



(a) **Correct Assumption** $d_i = 1$

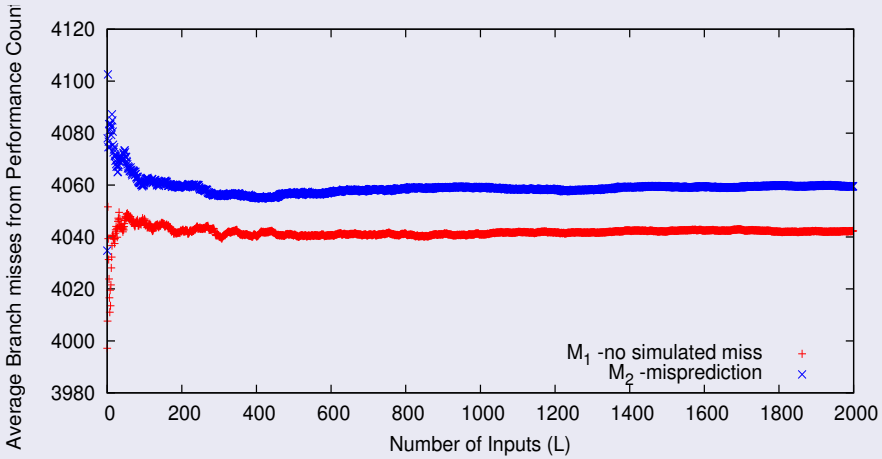


(b) **Incorrect Assumption** $d_i = 0$

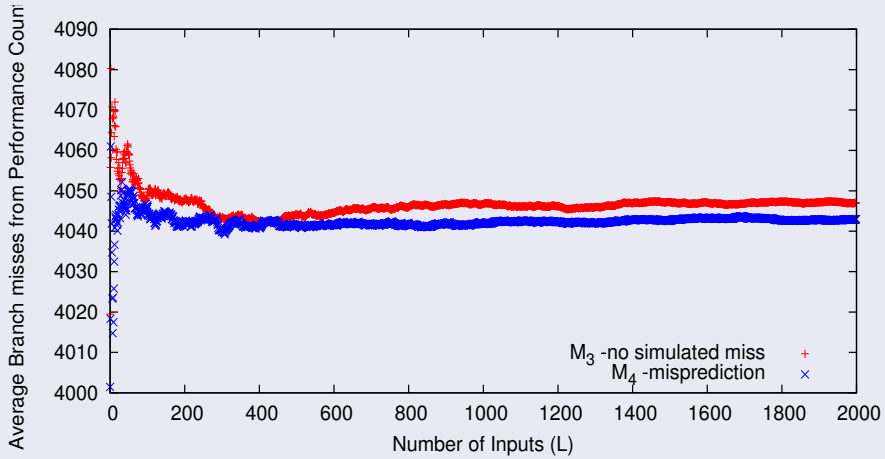
Figure: No identification of secret bit is possible using timing as side-channel with $L = 1000$ and $I = 1000$

Variation of parameters such as Number of Inputs (L) and Iteration (I)

Variation in separation with increase of Ciphertexts



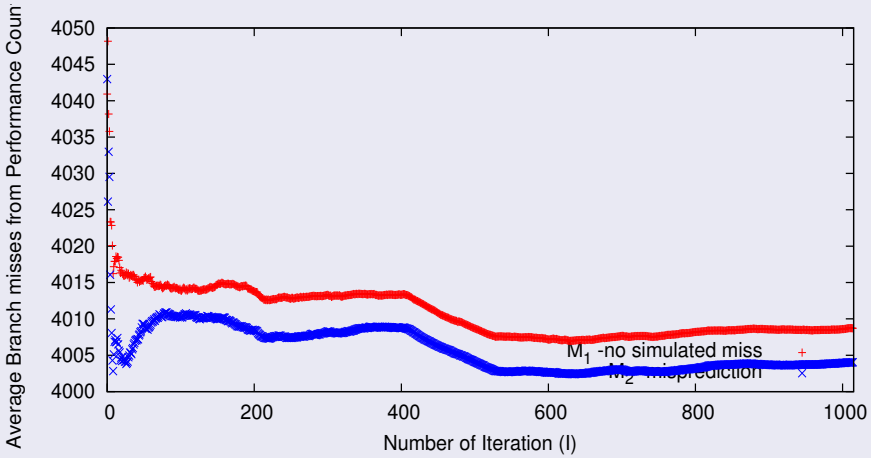
(a) **Correct Assumption** $d_i = 1$



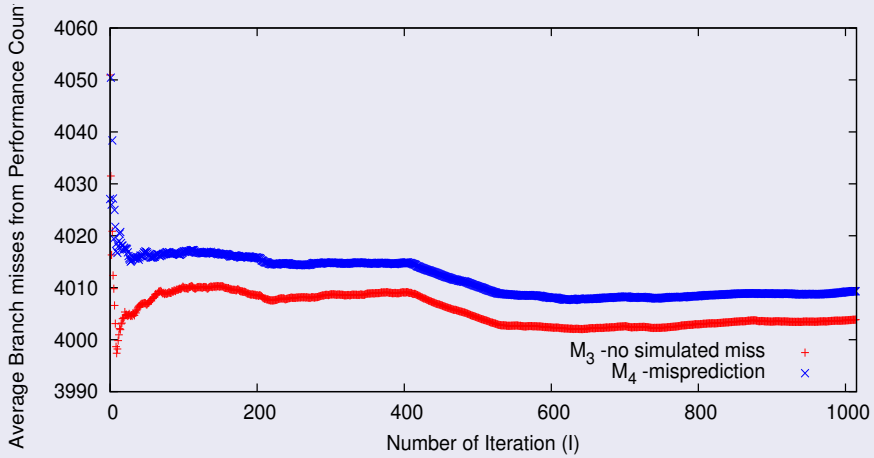
(b) **Incorrect Assumption** $d_i = 0$

Figure: Variation in the separation of branch misses for correct secret bit = 1 showing positive difference for M_1 and M_2 with the increase in number of ciphertexts(L), I = 100

Variation in separation with increase of Iterations



(a) **Incorrect Assumption** $d_i = 1$



(b) **Correct Assumption** $d_i = 0$

Figure: Variation in the separation of branch misses for correct secret bit = 0 showing positive difference for M_3, M_4 with the increase in number of iteration(I), L = 1000

RSA-OAEP Randomized Padding Scheme

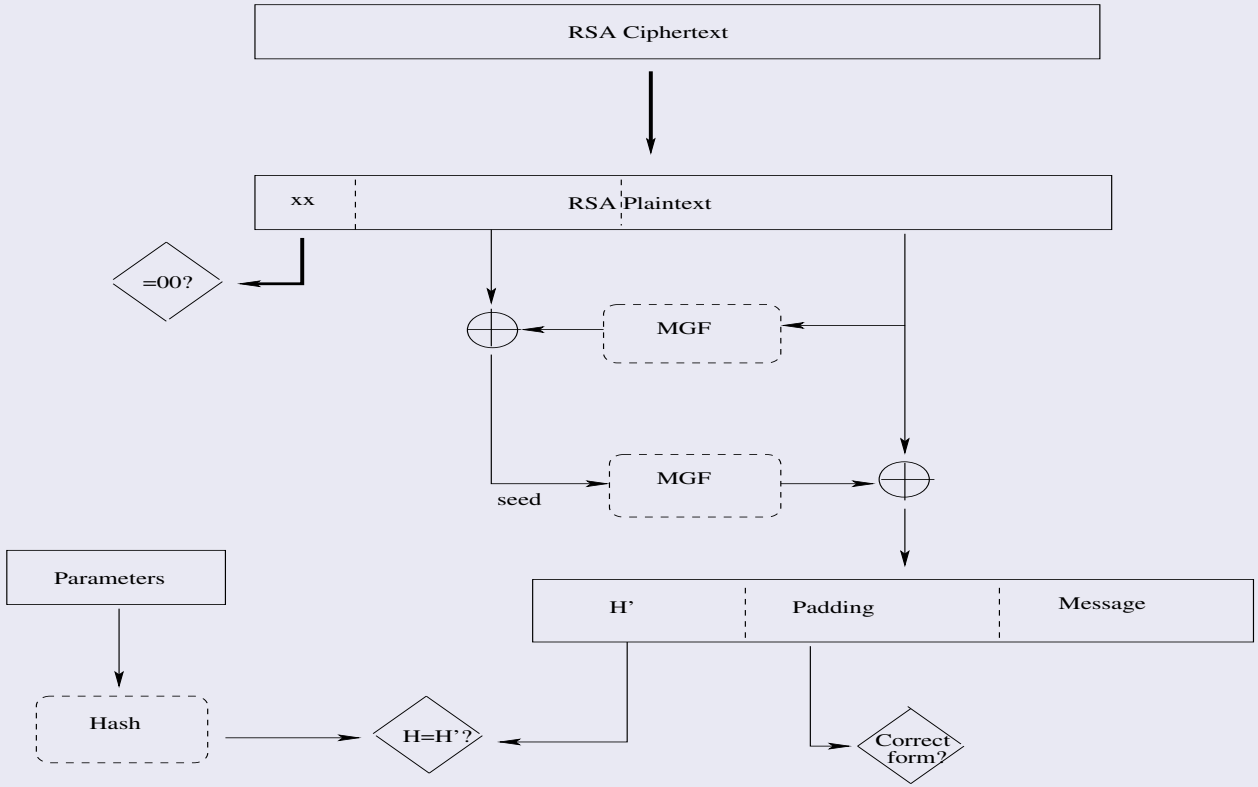
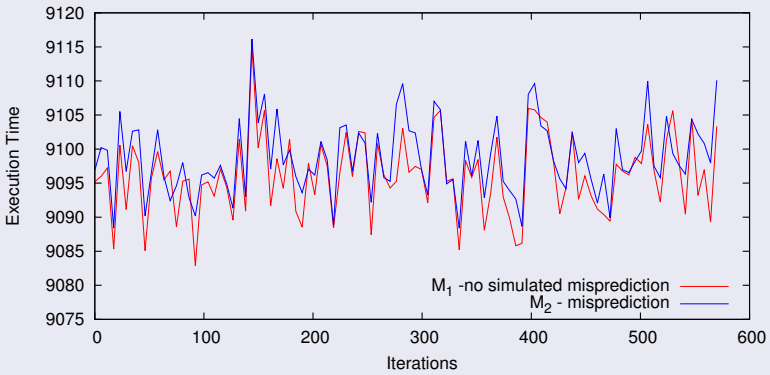
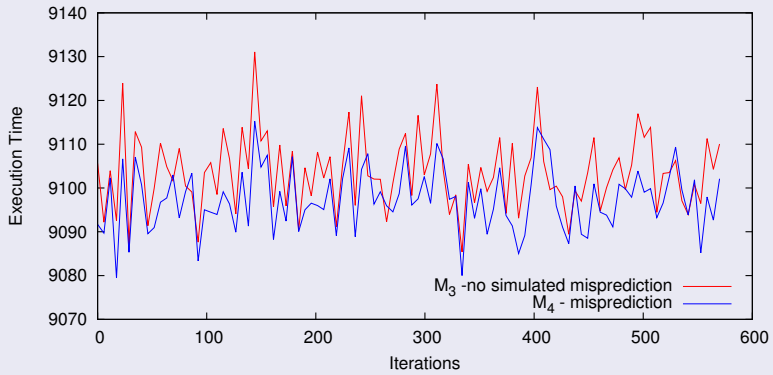


Figure: Decryption in RSA-OAEP procedure [14]

Separation for RSA-OAEP scheme



(a) Correct Assumption $d_i = 1$



(b) Incorrect Assumption $d_i = 0$

Figure: Branch misses from HPCs on RSA-OAEP implementation, correctly identifies secret bit $d_i = 1$, ciphertext set partitioned by simulated misses of bimodal predictor

Probable Countermeasures

- If input to MM algorithm is masked such that 2 random numbers r_1, r_2 generated at runtime and inputs are modified as $(a_r = a + r_1)$ and $(b_r = b + r_2)$, the branch predictor observes branches which depend on r_1, r_2. This masking strategy will prevent the adversary from simulating branch miss, since r_1, r_2 are randomly generated at run time.
- The effect of r_1, r_2 can be nullified by adding correction terms.
- There are other implementations of RSA, like CRT-RSA, can be more resistant, since the adversary cannot perform the necessary subsimulations (without knowing the prime factors of the RSA modulus).
- However, in presence of stronger fault models performance counters pose to be a threatening side channel.

Conclusion

- Experiments show that HPCs form threatening side-channel for existing implementations of RSA-like ciphers and similar implementations.

Conclusion

- Experiments show that HPCs form threatening side-channel for existing implementations of RSA-like ciphers and similar implementations.

Conclusion

- Experiments show that HPCs form threatening side-channel for existing implementations of RSA-like ciphers and similar implementations.
- The information provided by Performance Counters should be computed to access the performance, without providing a mechanism to extract secret information.

Conclusion

- Experiments show that HPCs form threatening side-channel for existing implementations of RSA-like ciphers and similar implementations.
- The information provided by Performance Counters should be computed to access the performance, without providing a mechanism to extract secret information.


W. Korn, P. J. Teller, and G. Castillo.

Just how accurate are performance counters?, pages 303–310. 2001.


Vincent M. Weaver and Sally A. McKee.

Can hardware performance counters be trusted?
In David Christie, Alan Lee, Onur Mutlu, and Benjamin G. Zorn, editors, *4th International Symposium on Workload Characterization (IISWC 2008), Seattle, Washington, USA, September 14-16, 2008*, pages 141–150. IEEE, 2008.


Leif Uhsadel, Andy Georges, and Ingrid Verbauwhede.

Exploiting hardware performance counters.
In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *FDTC*, pages 59–67. IEEE Computer Society, 2008.


John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore J. Stolfo.

On the feasibility of online malware detection with performance counters.
In Avi Mendelson, editor, *ISCA*, pages 559–570. ACM, 2013.


Xueyang Wang and Ramesh Karri.

Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters.
In *DAC*, page 79. ACM, 2013.


Kris Tiri, Onur Aciiçmez, Michael Neve, and Flemming Andersen.

An Analytical Model for Time-Driven Cache Attacks.
In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2007.


System Platforms Sector NTT DATA CORPORATION Tetsuo Takata, Platform Solutions Business Unit.

Perf for User Space Program Analysis.
[Online]. Available: `http://events.linuxfoundation.org/sites/events/files/lcjp13_takata.pdfhttp://events.linuxfoundation.org/sites/events/files/lcjp13_takata.pdf`, 2013.


September 2010 Arnaldo Carvalho de Melo, Linux Kongress.

The New Linux 'perf' tools.
[Online]. Available: `http://www.linux-kongress.org/2010/slides/lk2010-perf-acme.pdfhttp://www.linux-kongress.org/2010/slides/lk2010-perf-acme.pdf`, 2010.

Vincent M. Weaver and University of Maine.

Linux perf_event features and overhead.
In *2013 FastPath Workshop*, pages –, 2013.

Paul C. Kocher.

Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.
In Neal Koblitz, editor, *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, London, UK, 1996. Springer-Verlag.

Peter L. Montgomery.

Modular Multiplication without Trial Division.
Mathematics of Computation, 44(170):519–521, 1985.

Tse-Yu Yeh and Yale N. Patt.

Two-level adaptive training branch prediction.
In *MICRO*, pages 51–61, 1991.

Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert.

Predicting Secret Keys Via Branch Prediction.
In Masayuki Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2007.

James Manger.

A chosen ciphertext attack on rsa optimal asymmetric encryption padding (oaep) as standardized in pkcs 1 v2.0.
In *CRYPTO*, pages 230–238, 2001.