J Cryptogr Eng (2017) 7:299–310
DOI 10.1007/s13389-017-0165-6

SPECIAL SECTION ON PROOFS 2016

Formal fault analysis of branch predictors: attacking countermeasures of asymmetric key ciphers

Sarani Bhattacharya[1] · **Debdeep Mukhopadhyay**[1]

Received: 26 January 2017 / Accepted: 23 April 2017 / Published online: 9 May 2017

Abstract Implementations of asymmetric key algorithm have been threatened via timing side channels due to the behavior of the underlying branch predictors. However, the effect of faults on such predictors and the consequences thereof on the security of crypto-algorithms have not been studied. Motivated by the fact that unknown branch predictors of standard processors bear a strong correlation with 2-bit dynamic predictors, this paper develops a formal analysis of such a bimodal predictor under the effect of faults. Assuming a popular bit-flip fault model, the analysis shows that differences of branch misses under the effect of such faults can be exploited to attack implementations of RSA-like asymmetric key algorithms, based on square and multiplication operations. Furthermore, these attacks can be also threatening against Montgomery ladder of CRT-RSA (RSA implemented using Chinese Remainder Theorem) and even against fault attack countermeasures which stop or randomize the output in case of a fault. The theoretical claims have been substantiated by detailed fault simulations, where the difference of branch misses has been observed using the "perf" tool in Linux.

Keywords Fault attacks · Branch misses · HPCs · Branch prediction unit · Square and multiply algorithm · Montgomery ladder algorithm · RSA-CRT

✉ Sarani Bhattacharya
sarani.bhattacharya@cse.iitkgp.ernet.in

[1] Secured Embedded Architecture Laboratory (SEAL), Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal 721302, India

1 Introduction

Micro-architectural features are often captured by side channels which unknowingly leak the states of underlying hardware. Side-channel attacks allow malicious users to access sensitive data by monitoring power consumption, timing, or electro-magnetic radiation of the system. In modern microprocessors, there are several new sources of side channels and the performance improvement measures which are introduced, leak a significant amount of information. For example, hardware prefetchers for cache memories are shown to increase leakage in cache-timing attacks [7,17].

A leakage with respect to the branch mispredictions exists in the system where the instruction execution depends on an internal key. This is because the underlying branch predictors are subjected to instruction sequences conditioned on the key-dependent variables. Branch misprediction traces, a powerful side channel, can thus be observed by an adversary with respect to timing, power, or micro-architectural channels. There has been a few works in the literature which study branch misses with timing channels on real systems. In [2], four different types of timing attacks were performed on a standard RSA implementation exploiting the branch prediction unit by using both synchronous and asynchronous techniques. A further improved version of this attack [1,3] has also been carried out with proper knowledge of underlying hierarchical branch target buffer architecture of the target system. While in [6], the RSA crypto-system has been attacked where modular exponentiation has been implemented using both square and multiply and Montgomery ladder and the underlying multiplication and squaring has been implemented using Montgomery's method. This two-level branching has been targeted by the adversary to subsimulate the intermediate branching decisions for known key bits and known modulus N. But the attack in [6] does

not work on implementations like CRT-RSA since subsimulations are not possible in such algorithms. This is because the factors p, q of the RSA modulus N (which are used as the modulus in the two-step exponentiation in CRT-RSA) are not known to the adversary. Thus, the attack in [6] cannot be applied to CRT-RSA implementation. But the underlying exponentiation of CRT-RSA implementation still leaks information through the event branch misprediction. In the presence of faults, the difference of branch mispredictions observed over such exponentiations has the potential to leak the underlying secret key. The fault analysis attack proposed in this paper poses a potential threat on the contemporary systems, and such analysis on branch predictors has never been attempted in literature.

Fault analysis attacks are known to target the physical implementation of the cipher and tamper with the device to break the security mechanisms. The popular countermeasures to thwart the fault analysis attack randomize the cipher output on detection of the fault. The fault attack countermeasures attempt that the faulty output is obscured to any adversary. In this scenario, the processor's inbuilt special-purpose event counters such as the hardware performance counters (HPCs) could be a rich source of information leakage since they depict the internal state of the cipher execution in the form of various event counts. HPCs are a set of special-purpose registers to store the counts of hardware-related activities within the microprocessor. The HPC events can be monitored efficiently by user-space tools in spite of having fault attack countermeasures implemented in place. In this paper, we target event branch misses for implementations having key-dependent branching sequences.

Interestingly, the attack in [6] claims that though the underlying branch predictor is unknown; bimodal 2-bit predictor exhibits a very strong correlation with the actual statistics of branch misses obtained through HPCs. In this paper, we use the approximation to simulate branch mispredictions of branching sequences. In the presence of faults, this difference of mispredictions has been formalized leading to a characterization which can be utilized to launch a fault analysis attack.

The commonly used exponentiation algorithm like binary square and multiply in RSA-like public-key algorithms has unbalanced instruction execution which can be exploited by timing [13] and power side channels. On the contrary, if all the control flow paths have balanced instruction sequences as in Montgomery ladder algorithm [10], they are indistinguishable with respect to simple timing and power analysis. But still as these algorithms have key-dependent instructions, the branch miss event if monitored and analyzed properly could leak information.

In this paper, we formalize and characterize the difference of branch misses in the presence of bit-flip fault model. In the recent commercially available systems, inducing bit-flip in memory, are indeed a real threat due to the Rowhammer vulnerability [12]. The bit-flips can be triggered by repeated DRAM row activation, and recently, this vulnerability has been exploited to flip bits in secret exponent in memory [5]. In the presence of such faults, the difference of branch misses with the knowledge of the state of the bimodal predictor leads to unique key retrieval. The observations and the formalism have been utilized to develop an iterative key retrieval attack. Our attack model works even if the fault attack countermeasures are implemented, since the adversary is only concerned with the information regarding branch misses for the faulty execution. The major contributions of our paper are:

- The difference of branch misses observed through HPCs between the correct and the faulty execution can be modeled efficiently to develop a key recovery attack.
- We develop an iterative attack strategy, which simulates the branches corresponding to partially known exponent bits and observes the difference of branch misses from HPCs to reveal the next bit.
- The theoretical simulations are validated on secret key-dependent modular exponentiation algorithms as well as on CRT-RSA implementation.

The experimental validations are performed on the simulated fault model on several Intel platforms with 1024-bit key, and the results demonstrate that we are able to retrieve the secret bits uniquely.

The organization of the paper is as follows. The following Sect. 2 provides brief description on HPCs and vulnerability of HPCs in fault analysis attacks scenario. In Sect. 3, we formalize the differential of 2-bit predictor in fault attack setup. Section 4 establishes the formalism in context to the asymmetric key algorithms, and Sect. 4.2 explains the proposed attack algorithm. Section 4.4 provides the experimental validations for the attack strategy with simulated faults, and the final section contains conclusion of the work we present.

2 Differential analysis of the branch predictor

Branch misses are caused due to the branch predictors present in the underlying architecture. State machine of the 2-bit dynamic predictor in Fig. 1 has been extensively used as an underlying predictor in Intel family of microprocessors [9]. In real systems, the branch predictor hardware is much advanced and complicated compared to the above primitive model. The actual details of such architecture are not disclosed by the processor manufacturing companies. HPCs provide an interface to observe the number of branch misses from the underlying hardware.

Approximating a real-life branch predictor with a 2-bit dynamic predictor has been first demonstrated in [6]. This

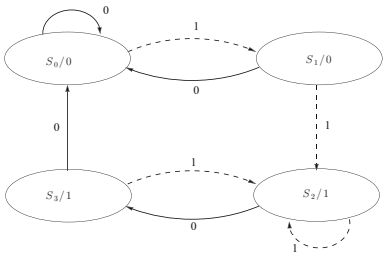

Fig. 1 Dynamic 2-bit predictor state machine

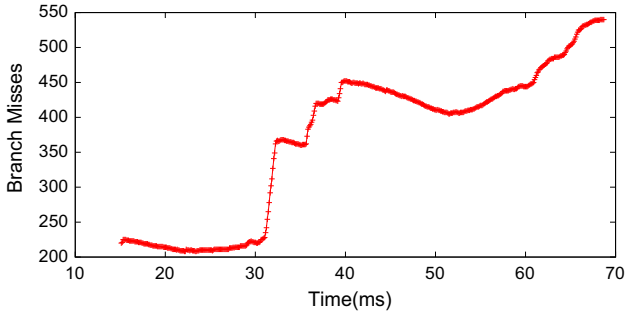

Fig. 2 Branch miss monitoring from an unprivileged user

approximation works successfully for our attack, because we perform a differential analysis. *While the actual value of the branch misses is not important, what is used in our attack is the relative ordering of branch misses of two processes. It may be stressed that while we develop the formal analysis using a simplistic model of the branch predictor, we later validate the attack and the claims thereof on an actual branch predictor of a real-life computing platform.* This paper focuses on the vulnerability of hardware event counts in fault analysis setups. HPCs being present in wide range of processors from embedded to standard desktop machines, exploiting secrets from a system by introducing fault is an unexplored area of research.

2.1 User accessibility of HPCs

HPCs contain rich source of information of the internal activities of the processor. The performance-optimizing tools monitor the HPCs and provide these information to processes or users in the aim to improve the performance of systems. Events, such as CPU cycles, cache misses, branch instructions, branch misses, page faults, context switches, are some of the many events whose counts are provided by the HPCs. On the occurrence of such hardware or software events the counter registers are incremented by one.

A handle on these HPCs can be easily obtained with the commonly used PC profiling tool "Perf" in Linux for statistical profiling of event "branch miss" from HPCs. In 2009, "perf" subsystem was added to the Linux kernel, and this makes user access to performance counters less clumsy, without kernel patches or re-compiles. The merge of perf event source with the main Linux kernel source tree has provided an easy access to the Linux users to hardware counters for the first time [20].

The values of the event counters which are available to user-level processes through Linux perf utility actually leaks a significant amount of information about the concurrently running processes in the system. We emphasize on this with a simple experimental scenario, where two users share the same hardware and have two different processes running: an unprivileged user running a multiplication operation and privileged user is performing exponentiation which is dominated mostly with conditional if-else statements. We have performed the experiments such that the unprivileged user observes Perf statistics for the user-level multiplication process concurrent to the privileged exponentiation process. As illustrated in Fig. 2, it has been observed on an Ubuntu 16.04 system that there is an unexpected increase in number of branch misses in the perf stat of the user process while the execution of exponentiation is performed. If the unprivileged user runs the multiplication process and observes the number of branch misses for the multiplication process, then it encounters around 200–220 branch misses. As in Fig. 2, we observe that the time from which there is an increase in the number of branch misses (as observed by the user process), coincides with the time when the exponentiation algorithm begins. This simple experiment shows how an unprivileged user process residing on same system as the privileged process can gain access to sensitive information of the privileged, or more generally any other user's process execution.

2.2 Effect of faults on HPCs and cryptographic implementations

When the secret key gets corrupted either due to memory corruptions or faulty computations in the key generation algorithm, the phenomenon manifests as a fault. However, a fault can also be introduced by skipping some target instructions [4]. Flips in the bit of conditional variables, often the keys itself, can lead to a branch being taken improperly. This wrong branching leads to instruction skips, thus having an equivalent effect on program executions. On platforms such as Xilinx Microblaze where the HPC accesses are provided [15], the instruction skip phenomenon can thus be exploited to create effects equivalent to bit-flips by altering the statistics of branching and thus leading to revelations of the secret keys.

In recent processors, Rowhammer is a term coined for disturbances observed in DRAM devices, where repeated row activation causes the DRAM cells to electrically interact within themselves [12,21]. These result in flipping bits [12] in DRAM due to discharging of the cells in the adjacent

rows. The authors in [12] show that Rowhammer vulnerability exists in majority of the recent commodity DRAM chips and is most prevalent for sub 40nm memory technology. Recently, authors in [5] have exploited this Rowhammer vulnerability to flip secret exponent bits residing in the memory of a x86 system. They provide a series of steps which can locate the secret in DRAM and repeated hammering induces bit-flips in the secret key. This motivates the study of differential analysis of HPCs when there are bit-flip faults.

In fault analysis attacks as well as their countermeasures, the adversary may be prevented in getting useful information, but the hardware events reflects the systems internal state which may have a dependence on the secret. The HPCs reveal the hardware event counts such as cache misses and branch misses, which may provide a runtime statistics to the adversary at the user-privilege level. For this reason, HPCs can be of potential threat with respect to fault analysis attacks and more notably against their countermeasures. In order to exploit the effect of event branch misses in situations of fault attack and their countermeasure, we target RSA-like ciphers, and in the fault attack scenario, we exploit the HPCs to reveal the secret exponents.

2.3 Case study for asymmetric key ciphers

In RSA-like crypto-systems, modular exponentiation is performed via square and multiply and Montgomery ladder implementations as illustrated in Algorithms 1 and 2. Though the Montgomery ladder algorithm prevents simple side-channel leakages, however, the instruction flow being conditioned on secret exponent leads to a leakage due to branch misprediction penalty.

Algorithm 1: Square and Multiply Algorithm

Input: y, x = $(x_{w-1}, x_{w-2}, \ldots, x_0)$, n
Output: $s = y^x \ (mod\ n)$
begin
  Let $s = 1$
  for $i = w - 1$ *down to* 0 **do**
    Let $s = s^2 \ mod\ n$.
    if $(x_i = 1)$ **then**
      Let $s = (s * y) \ mod\ n$.
    end
  end
  Return s.
end

The fault analysis attack presented in this paper has been extended for RSA-CRT implementations as well. The popular fault attack countermeasures for RSA-CRT include fault detection logic before the output is produced [8,11]. However, with such countermeasures implemented in place, the branch predictor observes branch statements over different key sequences during exponentiation and hence results in a different branch prediction statistics. In our fault attack, we show that the statistics can be observed via HPCs which can be exploited to reveal the secret key.

Algorithm 2: Montgomery Ladder Algorithm

Input: y, x = $(x_{w-1}, x_{w-2}, \ldots, x_0)$, n
Output: $s = y^x \ (mod\ n)$
begin
  Let $s_0 = 1, s_1 = y$.
  for $i = w - 1$ *down to* 0 **do**
    if $(x_i = 0)$ **then**
      Let $s_1 = s_0 * s_1, s_0 = (s_0)^2$.
    end
    else
      Let $s_0 = s_0 * s_1, s_1 = (s_1)^2$.
    end
  end
  Return s_0.
end

2.4 Targeting the *i*th secret bit

In public-key exponentiation algorithms as in Algorithms 1 and 2, the conditional multiplication statements depend on the secret key bits. In the presence of fault, the execution following the faulty exponent leads to a faulty output. Interestingly, we provide the relation of the exponent sequences and branch misses in brief. Let the n bit secret key be denoted as $(k_0, k_1, \ldots, k_i, \ldots, k_{n-1})$ and the fault induced key as $(k_0, k_1, \ldots, \overline{k_i}, \ldots, k_{n-1})$ differing only in the ith bit k_i. The multiplication statement of the square and multiply algorithm being conditioned on the secret key bits, the trace of taken or not-taken branches is conditioned on secret key bits and expressed as $(b_0, b_1, \ldots, b_{n-1})$.

- If a particular key bit say k_j is 1, then the conditional multiplication statement in the square and multiply algorithm 1 gets executed. Thus, the condition is checked first, and if the particular key bit is set then its immediate next statement, i.e., multiplication gets executed. Since this is a normal flow of execution, the branch is considered as not-taken, i.e., $b_j = 0$ in this case.
- While when $k_j = 0$, the multiplication statement is skipped and the execution continues with the next squaring statement. Thus, in this case branch is taken, i.e., $b_j = 1$.

2.5 Effect of compiler optimization options

In order to validate our understanding for conditional branching, we performed some experiments to observe the effect of optimization options in gcc on the conditional if-else structure of code. Similar to the exponentiation structure as in Montgomery ladder, we show an example of assembly generation for a simple conditional if-else code. The code prints "hello" if the "if" clause is true otherwise it prints "hi."

```
.LC3:
.string "hello"
.LC4:
.string "hi"
```

Table 1 Assembly generated using various optimization options in gcc

without Optimization	O1	O2	O3

.L5:
movl −36(%rbp), %eax
cltq
movzbl −32(%rbp,%rax), %eax
cmpb $49, %al
jne .L3
movl $.LC3, %edi
call puts
jmp .L4
.L3:
movl $.LC4, %edi
call puts
	.L5:
	cmpb $49, (%rsp,%rbx)
	jne .L3
	movl $.LC3, %edi
	call puts
	jmp .L4
	.L3:
	movl $.LC4, %edi
	call puts
		.L3:
		movl $.LC4, %edi
		call puts
		.L5:
		.....
		jne .L3
		movl $.LC3, %edi
		....
			.L3:
			movl $.LC4, %edi
			call puts
			.L5:
			...
			jne .L3
			movl $.LC3, %edi
			call puts
			...

So intuitively, if the "if" statement is true then the immediate next statement gets into the instruction pipeline, and thus, the branch is not-taken. On the contrary, if the "else" part is getting executed, then the branching is true and the branch statement is taken in such case. Table 1 shows the assembly translation of the code under various levels of optimizations. It is evident from all of these cases that, in spite of varying the optimization options, the conditional statement assembly remains the same.

3 Formalizing the differential of 2-bit predictor in faulty attack setup

In our work, we modeled the effect of the bimodal predictor to exploit the side-channel leakage of branch misses from the performance counters. In the following discussion, we characterize the differential of branch misses from correct and faulty branching sequences based on the behavior of 2-bit predictor. Various parameters used during the analysis are defined as follows: There is a sequence of n branches denoted as $(b_0, b_1, \ldots, b_{n-1})$ generated from execution of the algorithm under attack. A fault at the ith execution of the algorithm changes the branching decision for the ith instance.

The difference in branch misses (Δ_i) between the correct branching sequence $(b_0, b_1, \ldots, b_i, \ldots, b_{n-1})$ and the faulty sequence $(b_0, b_1, \ldots, \overline{b_i}, \ldots, b_{n-1})$ simulated theoretically over a 2-bit predictor algorithm can be at least -3 and at most 3.[1] The simulated difference for any branch sequence will always take a value in the range $[-3, 3]$, and this depends on the subsequent branching sequences starting from the ith flipped branch. This effectively means that the branching decision for the ith instance have a great impact in determining the range of Δ_i.

In the following discussion, we will analyze the difference in branch misses Δ_i depending on the branching decision b_i. All symbols used through out our analysis are detailed in Table 2. Let K denote the actual branching sequence and F_i denote the faulty branch sequence differing from the actual at the ith branching instance. The state of the 2-bit predictor after encountering j branches is St_j^K and for the faulty sequence is $St_j^{F_i}$. The 2-bit predictor predicts for the branch statement corresponding to the $(j+1)$th instance as P_{j+1}^K on the basis of previous j branch decisions for the correct sequence and similarly for the faulty sequence as $P_{j+1}^{F_i}$. We state 2 properties using the state transition of the 2 bit dynamic predictor.

Table 2 Tabular representation of symbols

Symbols	Meanings with respect to their analysis
$(b_0, b_1, \ldots, b_{i-1})$	Sequence of taken or not-taken known branches
St_j^K	State of 2-bit predictor after j conditional branches with respect to the correct sequence
$St_j^{F_i}$	State of 2-bit predictor after j conditional branches with respect to the faulty sequence
P_{j+1}^K	Branch predicted by 2-bit predictor for branch statement corresponding to $(j+1)$th bit of correct sequence
$P_{j+1}^{F_i}$	Branch predicted by 2-bit predictor for branch statement corresponding to $(j+1)$th bit of faulty sequence

Property 1 **If** $St_{i-1}^K = S_0$ **or** $St_{i-1}^K = S_2$, **then** $P_i^K = P_i^F = b_{i-1}$.

– This follows from the fact that the fault is only in the ith branch, and thus, there is no effect till this instance. Also, from Fig. 1, for states S_0&S_2, the branch statement for last input branch, i.e., the $(i-1)$th branch is same as the ith predicted branch.

Property 2 **If** $St_{i-1}^K = S_0$ **or** $St_{i-1}^K = S_2$, **then there are guaranteed mispredictions for branch statement at the** ith **instance for either** K **or** F_i. **If the branch statement corresponding to** $(i+1)$th **instance is not same as the predicted** P_i^K, **then there is a mismatch between the correct**

[1] We explain the upper bound of 3 at the end of this section. The lower bound can be similarly established.

and the faulty sequence in the predictor's output for the $(i+2)$th **position as** $P^K_{i+2} \neq P^{F_i}_{i+2}$.

– Either b_i or $\overline{b_i}$ results in a misprediction for the ith branch of the correct or the faulty sequence. Since $b_{i-1} = P^K_i$, at least two mispredictions are required at i, $(i+1)$th position to make $P^K_{i+2} \neq P^{F_i}_{i+2}$. Thus, if $b_{i+1} \neq P^K_{i+1}$, then there will be two consecutive mispredictions at i, $(i+1)$th position either for the secret or the faulty sequence resulting in $P^K_{i+2} \neq P^{F_i}_{i+2}$.

Using the stated properties, we formally model the probable values for simulated differential Δ_i given the ith branching decision b_i. In the following discussion, we provide an analysis for $St^K_{i-1} = S_0$,

1. **If** $St^K_{i-1} = S_0$ **and** $b_i = 0$ **then** $\Delta_i \in \{0, 1, 2, 3\}$
   – $P^K_i = P^F_i$, since every predicted branches from 2-bit branch predictor for 0th to ith bit are same for both correct and faulty sequence. By Property 1, since the state is either S_0/S_2 the $(i-1)$th, input bit is same as the predicted branch for the ith bit, $b_{i-1} = P^K_i = P^F_i = 0$.

   – **Case 1(a): The** ith **branching decision** b_i **is chosen to be** 0,

   – Thus, for the correct sequence, ith branch is same as the predicted branch $b_i = P^K_i = 0$, and branch miss does not increase.
   – Again for the correct sequence, predictor on getting bit $b_i = P^K_i$, predicts $P^K_{i+1} = P^K_i = 0$ same as the previous predicted branch value.

   – **Case 1(b): On the contrary, for** ith **bit of the faulty sequence**

   – $\overline{b_i} = 1 \neq P^F_i$, and branch miss increases by 1.
   – Again, since $b_{i-1} = 0 \neq \overline{b_i}$ and $\overline{b_i} \neq P^{F_i}_i$, there is a single misprediction for the ith bit. A single misprediction is not enough to cause the mismatch in the predicted branch output of the correct and the faulty sequence bits. Thus, the predicted branch for the $i+1$th bit for the faulty sequence does not change as $P^F_{i+1} = 0 = P^F_i = b_i$. This makes the intermediate difference in branch misses upto the ith bit as 1.

   –**Case 1(c): If** $b_{i+1} = 0$,

   – For the $i+1$th bit, if and only if $b_{i+1} = 0$, $(i+1)$th predicted branch $P^K_{i+1} = P^F_{i+1} = b_{i+1} = 0$, and the further changes in branch miss would not get reflected in their difference.
   – Thus, if $St_{i-1} = S_0$ and $(b_i, b_{i+1}) = (0, 0)$, then $\Delta_i = 1$.

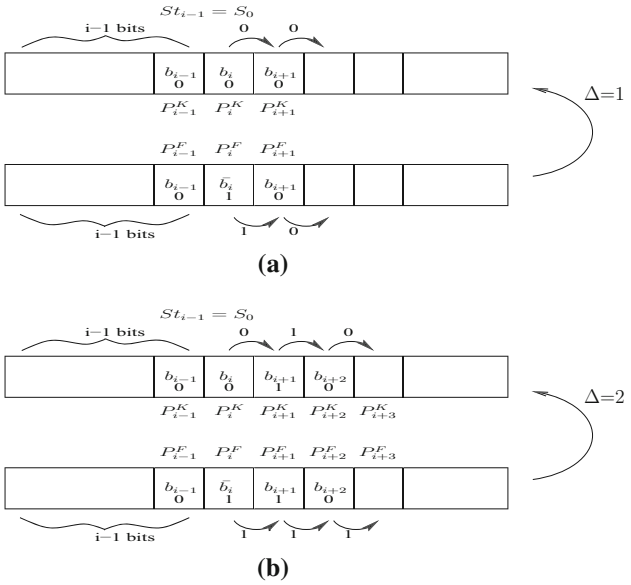

Fig. 3 Variation of simulated branch misses on the ith branching decision having $St_{i-1} = S_0$. **a** $\Delta_i = 1$, **b** $\Delta_i = 2$

The sequence of the subsequent bits when $\Delta_i = 1$ is depicted in Fig. 3a.

–**Case 1(d): Again if** $b_{i+1} = 1$,

– if $b_{i+1} = 1 = \overline{b_i}$, then branch miss for both sequences increases by 1 and does not add up to the difference.
– but by Property 2, for the two consecutive mispredictions of faulty key at the $i, i+1$th position, $(\overline{b_i} = b_{i+1} = 1 \neq P^F_{i+1} = 0 = P^K_{i+1})$ prediction for the correct and the faulty sequence for the $(i+2)$th branch differs as $P^F_{i+2} = 1 \neq P^F_{i+1} = P^K_{i+2} = 0$.

– **Case 1(e): If we assume** $b_{i+2} = P^K_{i+2} = 0$, so there is no increase in branch misses for correct sequence. While $P^F_{i+2} = 1 \neq b_{i+2} = 0$, increasing intermediate difference of branch miss to 2.

– Thus, if $St_{i-1} = S_0$ and $(b_i, b_{i+1}, b_{i+2}) = (0, 1, 0)$, then the intermediate difference in branch miss becomes 2.
– And for this case, $P^K_{i+3} = 0 \neq P^F_{i+3} = 1$.

The scenario where the differential of bit sequences $\Delta_i = 2$ is shown in Fig. 3b.

– **Case 1(f): On the other hand if** $b_{i+2} = 1$,

– $b_{i+2} = 1 \neq P^K_{i+2} = 0$, so there is an increase in branch misses for correct sequence.
– But for the faulty sequence $P^F_{i+2} = 1 = b_{i+2}$, thus the intermediate difference becomes 0.

Figure 4a illustrates the scenario when $\Delta_i = 0$.

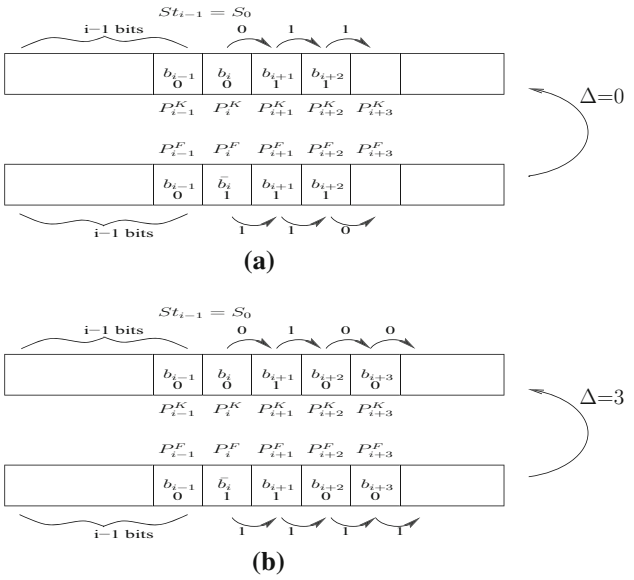

Fig. 4 Variation of simulated branch misses on the ith branching decision having $St_{i-1} = S_0$. **a** $\Delta_i = 0$, **b** maximum variation

– **Case 1(g): Considering the case,** $(b_i, b_{i+1}, b_{i+2}) = (0, 1, 0)$, and if $b_{i+3} = 0$

- $b_{i+3} = P^K_{i+3} = 0$, so there is no increase in branch misses for correct sequence.
- While $P^F_{i+3} = 1 \neq b_{i+3} = 0$, increasing the intermediate difference of branch miss to 3.
- Thus, for $(b_i, b_{i+1}, b_{i+2}, b_{i+3}) = (0, 1, 0, 0)$ the difference in branch miss $\Delta_i = 3$

This is the maximum difference in branch miss that can be obtained as in Fig. 4b. Since for the $(i + 4)$th bit for correct and faulty sequence,

- Finally, for the $(i + 4)$th branch, the predicted value for faulty sequence flips, $P^F_{i+4} \neq P^F_{i+2}$
- Thus, for both for the correct sequence and the faulty sequence, the predicted value becomes equal as $P^K_{i+4} = P^F_{i+4}$, the next bits being identical for both the sequence, has no effect in changing the difference value.

In all of the above cases, we assumed that $b_i = 0$, and for all of these cases, the simulated difference can be either 0 or $+ve$. Thus, we state that, if $St^K_{i-1} = S_0$ and $b_i = 0$ then $\Delta_i \in \{0, 1, 2, 3\}$.

2. **If** $St^K_{i-1} = S_0$ **and** $b_i = 1$ **then** $\Delta_i \in \{0, -1, -2, -3\}$
Similar to the previous analysis,
– $P^K_i = P^F_i$, since every predicted branches from 2-bit branch predictor for 0th to ith bit are same for both correct and faulty sequence.
– By Property 1, since the state is $St^K_{i-1} = S_0$, the $(i-1)$th input bit is same as the predicted branch for the ith bit, $b_{i-1} = P^K_i = P^F_i = 0$.

– **Case 2(a): Now, the** ith **branching decision** b_i **is chosen to be** 1,

- Thus, for the correct sequence, ith branch is not same as the predicted branch $b_i = 1 \neq P^K_i = 0$, and branch miss increases by 1.
- Again, for the correct sequence, predictor on getting a single misprediction bit $b_i \neq P^K_i$, also predicts $P^K_{i+1} = P^K_i = 0$ same as the previous predicted branch value, since a single misprediction is not enough to change the prediction output.

– **Case 2(b): On the contrary, for** ith **bit of the faulty sequence**

- $\overline{b_i} = 0 = P^F_i$, and branch miss does not increase.
- Again, since $b_{i-1} = 0 = \overline{b_i}$ and $\overline{b_i} = P^{F_i}_i$, there is a no misprediction for the ith bit. Thus, the predicted branch for the $i + 1$th bit for the faulty sequence does not change as $P^F_{i+1} = 0 = P^F_i = \overline{b_i}$. This makes the intermediate difference in branch misses upto the ith bit as -1.

The analysis is apparent for $St^K_{i-1} = S_0$ and $b_i = 1$ having $\Delta_i = -1$. This analysis can be extended using the previous reasoning and the fact can be similarly established that if $St^K_{i-1} = S_0$ and $b_i = 1$ then the simulated differential either becomes negative or at most 0 and can never be greater than 0. Thus, reconstructing the steps in the previous analysis, we can demonstrate that if $St^K_{i-1} = S_0$ and $b_i = 1$ then $\Delta_i \in \{0, -1, -2, -3\}$. Alternatively, an analysis can be provided for $St_{i-1} = S_2$ for the ith branching decision b_i, where

- $P^K_i = P^F_i = 1$, since $St_{i-1} = S_2$ and every predicted branches from 2-bit branch predictor for 0th to ith bit are same for both correct and faulty sequence.
- By Property 1, since the state is $St^K_{i-1} = S_2$ the $(i - 1)$th input bit is same as the predicted branch for the ith bit, $b_{i-1} = 1 = P^K_i = P^F_i = 1$.

For $St_{i-1} = S_2$, the similar analysis as in $St_{i-1} = S_0$ can be reconstructed and modeled in exact same steps, and in this case, we can state that,

3. **If** $St^K_{i-1} = S_2$ **and** $b_i = 0$ **then** $\Delta_i \in \{0, -1, -2, -3\}$, and
4. **If** $St^K_{i-1} = S_2$ **and** $b_i = 1$ **then** $\Delta_i \in \{0, 1, 2, 3\}$

The above analysis shows that mispredictions simulated over the branching sequences from the 2-bit predictor shows a unique classification of Δ_i values when classified based on ith branching decision. In the following subsection, we

observe the similar classification using branch mispredictions from HPCs. For developing an iterative algorithm to recover the key, the differential statistics obtained by monitoring the HPC should be according to the analysis presented in Sect. 3. We simulate the fault model in software and subsequently utilize it to perform a fault analysis in the next section.

4 Implication of fault in the key for asymmetric key algorithms

In this section, we aim to analyze observations using data from HPCs on exponentiation algorithms depending on the ith branching decision. In the subsequent discussion, we describe the entire analysis in the presence of a fault model on public-key exponentiation algorithms of key length n bits.

4.1 Fault model

The fault model we assume in the subsequent work is that the adversary is capable of introducing a bit-flip fault in the secret keystream. This **transient fault** flips the target bit of the key only for the current computation in the system, but when the execution is repeated the fault vanishes, and the subsequent executions proceed with the original secret key. The adversary follows a "bit-flip model" which means that if a key bit at ith position is 1 then it flips to 0, likewise if 0 then becomes 1. Thus, the branching sequence can be eventually modified by introducing a fault at the ith bit position of the key. This has indeed become more relevant with the recent practical fault model in contemporary x86 systems supporting DDR3 DRAM modules due to the recent threat of the Rowhammer attacks. The Rowhammer attacks flipping bits in memory are powerful in causing privilege escalation [18], changing control flow of ongoing processes [19] and also inducing bit-flip in a secret exponent residing in memory [5]. Also, as stated in Sect. 2 the effect of altering branching decision can be alternatively achieved with an instruction skip model. In comparison with the bit-flip model, the instruction skip paradigm is often more practical in embedded soft-core processors like Xilinx Microblaze [16].

4.1.1 Differential behavior of HPC due to an ith bit fault

We consider a scenario, where there are a large number of candidate keys and their faulty counterparts. The secret and faulty sequences only differ at the ith bit, the previous 0th to $(i-1)$th bits being same for both the exponents, the branch sequences corresponding to secret and its faulty counterpart varies only at the ith bit. The following observation is made when both the secret and faulty key sequences have branching decisions such that after simulating 0th to $i-1$th branch the predictor state is at S_0. At this point, there may be two possible values of ith bit.

- If the ith bit is 0, then the branch corresponding to this bit is taken (1) and alternatively for the faulty exponent this results in a not-taken (0) branch (because of the flip at the ith position).
- Again if the ith secret bit is 1, then the secret exponent branch is not-taken (0) while the branch for the faulty exponent is taken (1).

Following the formalism in Sect. 3, the $\Delta_i \geq 0$ for $b_i = 0$ and $\Delta_i \leq 0$ for $b_i = 1$ and the difference of simulated branch misses range from $(-3, 3)$. While validating this formal analysis with the actual event counts from the HPCs, the exponentiation operations are run on both the correct sequence and the faulty sequences on an Intel i5 system running Ubuntu 14.04 LTS. The branch miss counts as reported from the perf tool are noted.

In the first step, the number of branch misses for exponentiation operation is observed using the secret exponent from the HPCs. In the next step, exponentiation is performed over the faulty sequence, simultaneously observing the number of branch misses from HPCs. The difference of branch misses obtained through HPCs is denoted as δ_i. Figure 5a is plotted with differences of branch mispredictions from HPCs (δ_i) and two different colors highlight the behavior of two separate distribution based on the ith branching decision of the secret exponent assuming the state of the 2-bit dynamic predictor for the known $i-1$ branches is S_0. From the figure, it is apparent that if the ith branch of the secret key is not-taken then the difference has a strict positive bias. Alternatively, if the ith branch of the key is taken, then the difference of branch misses has a negative bias. Formally, if $St_{i-1}^{K} = S_0$,

- If $b_i = 0$, then $\delta_i > 0$
- Else if $b_i = 1$, then $\delta_i < 0$

The values of branch misses in Fig. 5 are observed using the perf utility in Linux. The differences as appears in the figure ranges much higher than the range we simulated. This feature can be attributed due to the background processes running, since the values reported by perf is naturally affected by various other processes running in the system other than the exponentiations. However, the nature of the variation of the branch miss events reported by the performance counters conforms with the developed theory.

A similar observation is made with $St_{i-1}^{K} = S_2$, and Fig. 5b is plotted for ith branching decision with the state of the 2-bit dynamic predictor for the known $i-1$ branches is S_2. Alternative to the previous observation, if ith branch of the secret key is not-taken then the difference has a strict negative bias and if the ith branch of the secret key is taken then the difference of branch misses has a positive bias. Precisely, if $St_{i-1}^{K} = S_2$,

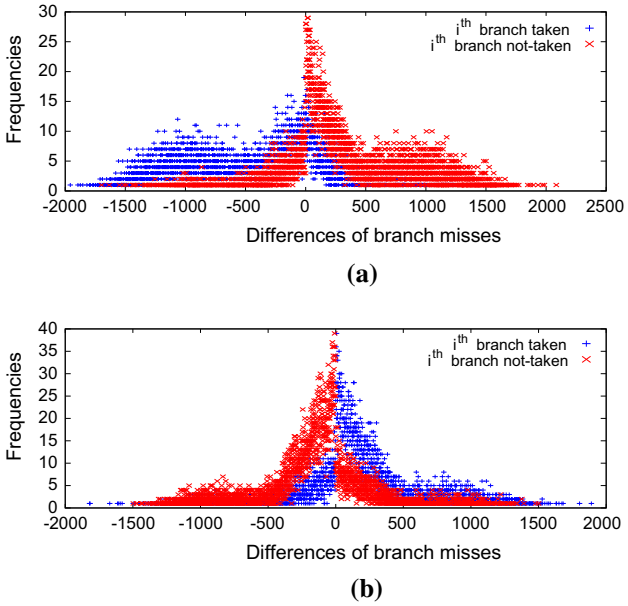

Fig. 5 Variation of branch misses from performance counters based on the ith branching decision. **a** When $St_{i-1}^{K} = S_0$, **b** $St_{i-1}^{K} = S_2$

– If $b_i = 0$, then $\delta_i < 0$
– Else if $b_i = 1$, then $\delta_i > 0$

Thus, we conclude that the properties of differential of branch misses obtained by theoretical simulation also holds for the statistics observed through HPCs.

4.2 Developing the attack algorithm

The observations regarding the ith branching decision illustrated in previous Sect. 3 can be efficiently modeled in the form of an attack algorithm. The attack algorithm to reveal secret of length n bits is developed on the basis of the following assumptions:

1. The bit-length n of the key is known to the adversary.
2. The adversary starts from the most significant bit (always 1), recovering the subsequent bits one bit at a time.
3. A fault introduced at the ith position of the key sequence eventually flips the ith bit of the key.
4. The already retrieved $i-1$ bits can be simulated by using the 2-bit predictor algorithm and reveals the state $St_{i-1} \in \{S_0, S_1, S_2, S_3\}$.
5. The difference in branch misses (δ_i) for the secret key ($\boldsymbol{K}$) and the faulty key ($\boldsymbol{F}_i$) is monitored by executing exponentiation algorithms on actual systems.

The adversary attack model observes the difference of branch misses from the secret key and its corresponding faulty key and gradually retrieves one unknown key bit at a time from the most significant bit (left) to the least significant bit (right). To determine the ith bit value from the left, the adversary already has the knowledge of the previous $i-1$ bits. Following the 2-bit predictor algorithm, $i-1$ branch transitions over the states S_0, S_1, S_2, S_3 can be traced by the adversary.

Let δ_i be the differences of branch misses over the secret and faulty exponent observed from the HPCs. The state St_{i-1}^{K} after $i-1$ transitions on the partially known key bits can be determined by simulating over the 2-bit dynamic predictor. We determine the next bit nb_i as,

1. If $St_{i-1}^{K} = S_0/S_2$:
   – If $\delta_i < 0$,
     – $nb_i = 0$, if $St_{i-1}^{K} = S_2$ and
     – $nb_i = 1$, when $St_{i-1}^{K} = S_0$.
   – Else if $\delta_i > 0$
     – $nb_i = 0$, if $St_{i-1}^{K} = S_0$ and
     – $nb_i = 1$, when $St_{i-1}^{K} = S_2$.
2. Else if, $St_{i-1}^{K} = S_1/S_3$:
   For states S_1 and S_3, if we flip the $(i-1)$th bit, the state upto $(i-1)$th bit changes to S_0 or S_2. Now, following the classification as described for S_0 and S_2 the difference can uniquely reveal the ith secret bit. So a solution which already appears for S_0 or S_2 can be adopted for $St_{i-1} = S_1/S_3$ since it will make the attack algorithm easier.
   – the characteristic property for $St_{i-1} = S_1/S_3$ is such that $b_{i-2} = P_{i-1} = P_i \neq b_{i-1}$.

   If we inject a fault at $(i-1)$th position, then branching decision b_{i-1} gets complemented. Effectively, if $St_{i-1}^{K} = S_1$ previously then after fault $St_{i-1}^{F_{i-1}}$ becomes S_0. Similarly, if $St_{i-1}^{K} = S_3$ previously then after fault $St_{i-1}^{F_{i-1}}$ becomes S_2.
   Let $\delta_{i-1,i}$ be the differences of branch misses over the faulty exponents observed from the HPCs. We determine the next bit nb_i as,
   – If $\delta_{i-1,i} < 0$,
     – $nb_i = 0$, if $St_{i-1}^{K} = S_3$ and
     – $nb_i = 1$, when $St_{i-1}^{K} = S_1$.
   – Else if $\delta_{i-1,i} > 0$
     – $nb_i = 0$, if $St_{i-1}^{K} = S_1$ and
     – $nb_i = 1$, when $St_{i-1}^{K} = S_3$.

Before starting the actual validation of above attack algorithm, we analyze the branch misprediction statistics generated by HPCs on the exponentiation algorithms.

4.3 Modeling the system noise

The modeling of data generated by the HPCs is purely implementation dependent. The algorithm under consideration must have key-dependent execution sequences which are captured by the HPCs while execution is performed on an

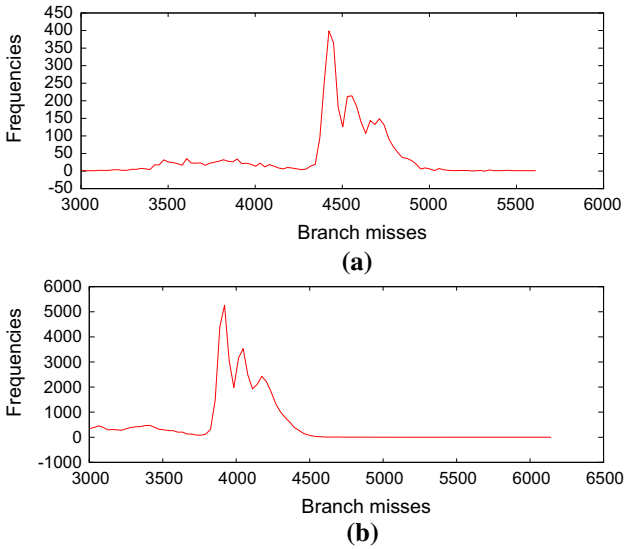

Fig. 6 Variation of branch misses from performance counters. **a** Due to exponentiation on secret exponent, **b** due to environmental processes running in the system

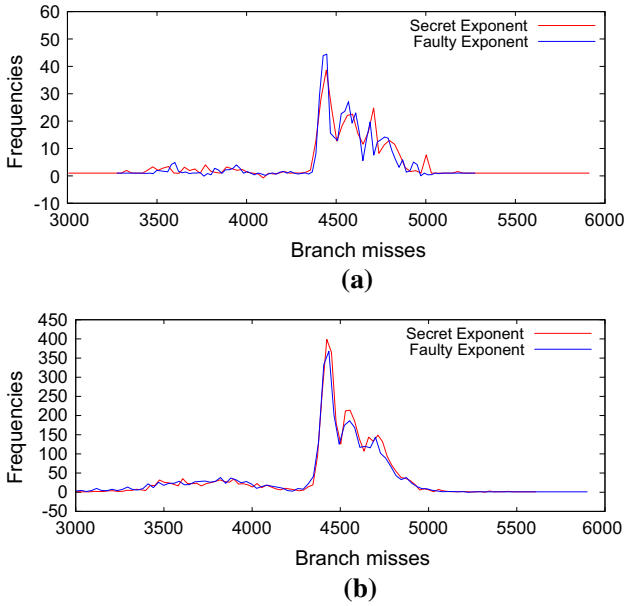

Fig. 7 Distribution of branch misses of secret and faulty exponent on square and multiply implementation from HPCs having $St_{i-1} = S_0$. **a** $b_i = 0$ and $\delta_i = 14.014$, **b** $b_i = 1$ and $b_i = 1$ and $\delta_i = -35.79$

input. We considered square and multiply and Montgomery ladder algorithmic implementations running on a sample of inputs, and the distribution of branch misses generated shows an interesting pattern. Branch misprediction observed from the HPCs over exponentiations on the secret exponent bits over 10000 randomly generated inputs can be observed as a mixture of multiple Gaussian distributions. One such distribution of branch misses is shown in Fig. 6a. The distribution can be observed as a combination of three Gaussian distributions having three separate statistics.

A similar experiment is carried out, but this time the conditional multiplication statement in the exponentiation is removed from the code. This effectively means that the recent executable does not posses any conditional statement dependent on the exponent bit values. Branch mispredictions are also monitored for this executable, but in this scenario since there are no conditional statement in the executable, the branch mispredictions accounted by the HPCs are due to the environmental processes running in the system. One such distribution is shown in Fig. 6b. Interestingly, this distribution of branch misprediction due to the environmental processes also constitutes of three Gaussian distributions. The reason for this three overlapping Gaussian distributions may be accounted by the Operating System policies in multitasking systems. Several environmental processes running in the system are responsible for this environmental branch mispredictions (which are not due to the target executable for which they are accounted). Figure 6a shows similar nature to this noise distribution with a shift in the respective statistics with an increase in branch misprediction due to the conditional statements from the secret exponents.

4.4 Validation of the attack algorithm

We present the validation of previous discussion through experiments on RSA algorithm, the exponents being 1024 bits. The fault model is simulated on software and the experiments target data from HPCs on various Intel processors like Intel Core-2 Duo E7400, Intel Core i3 M350 and Intel Core i5-3470. Initially, the attack is performed on the square and multiply exponentiation implementation. Figure 7a illustrates the distributions of branch misses from the square and multiply exponentiation over the secret and faulty exponent having $St_{i-1} = S_0$ for $b_i = 0$ and the fault being introduced at $i = 1019$th position. The distributions in Fig. 7a have the mean difference $\delta_i = 14.014$. Since $St_{i-1} = S_0$, and with positive value of δ_i, according to the Attack Algorithm the next branch is decided as $nb_i = 0$ which actually matches with correct branch $b_i = 0$, thus the secret exponent is uniquely retrieved as $k_i = \overline{b_i} = 1$. Similarly, Fig. 7b illustrates the distribution of branch misses where we target $i = 548$th location of the same exponent having branch taken, i.e., $b_i = 1$ and $St_{i-1} = S_0$. The mean difference of branch misses observed from HPCs are $\delta_i = -35.79$ which correctly decides the ith branch to be 1.

The above figures are the distributions of branch misses from the secret and its faulty counterpart for a single randomly chosen target location. This can be easily performed on all subsequent bits to recover all 1024 bits of RSA exponent.

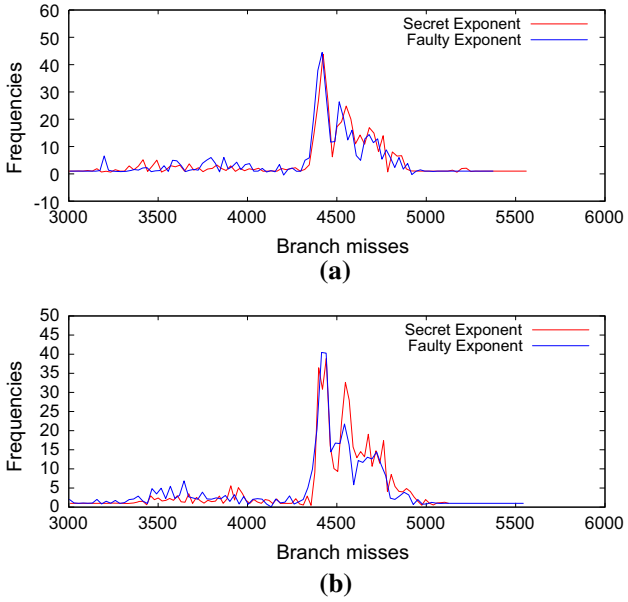

Fig. 8 Distribution of branch misses of secret and faulty exponent on Montgomery ladder implementation from HPCs having $St_{i-1} = S_0$. **a** $b_i = 0$ and $\delta_i = 9.828$, **b** $b_i = 1$ and $\delta_i = -139.086$

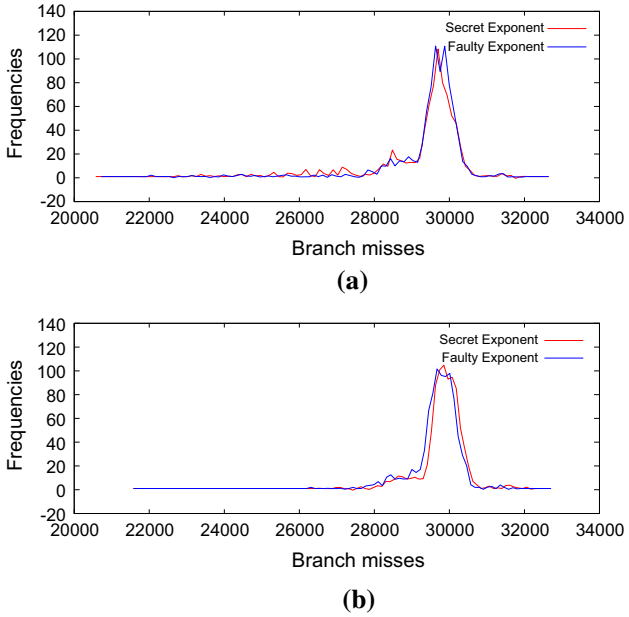

Fig. 9 Distribution of branch misses of secret and faulty exponent on CRT-RSA implementation from HPCs having $St_{i-1} = S_0$. **a** $d_{p_i} = 0$ and $\delta_i = 243.212$, **b** $d_{p_i} = 1$ and $\delta_i = -136.029$

4.4.1 Attacks on Montgomery ladder algorithm

A popular countermeasure of simple side-channel leakages from the unbalanced instruction execution of square and multiply algorithm is the Montgomery ladder implementation as in Algorithm 2. This algorithm is having balanced instructions conditioned on the secret key bits. The differential fault analysis for square and multiply similarly holds for Montgomery ladder implementation. Figure 8a, b illustrates the differential by using data from performance counters and using Montgomery ladder exponentiation as the underlying exponentiation algorithm. Figure 8a illustrates a situation having $k_i = 1$ for $i = 248$ where $St_{i-1} = S_0$, $b_i = 0$ and the branch misses from HPCs $\delta_i = 9.828$ reveals a positive difference correctly identifying $nb_i = 0$, thus correctly identifying the secret exponent bit. While Fig. 8b shows a negative difference, $\delta_i = -139.086$ correctly identifying $k_1 = 0$ for $i = 337$.

4.4.2 Attacks on CRT-RSA implementation

Experimentations on CRT-RSA also reveal that the attack algorithm can successfully identify the exponent bits. Similar to the square and multiply and Montgomery ladder algorithms, Fig. 9a, b shows illustrations of two instances of the CRT-RSA implementation with square and multiply implementation where the fault is induced in d_p, while exponentiation for d_q is computed unaffected. In both situations, the target exponent bits of d_p are shown to be retrieved correctly and uniquely. Similarly, exponent bits for d_q can also be retrieved. A probabilistic poly$(\log N)$ time algorithm to factorize Modulus N [14] on the basis of the knowledge of N, e, d_p, d_q already exists which can factorize N in probabilistic poly$(\log N)$ time.

The popular fault attack countermeasures of the RSA-CRT algorithm performs a comparison of the correct computation with a faulty computation. So in all of the countermeasures while the exponentiation operation is being performed on the secret as well as the faulty exponent, the branch miss events recorded in the performance counters can be potentially used to reveal secret exponents d_p and d_q. Thus, the fault attack featuring differences of branch misses by modeling the 2-bit predictor algorithm can be utilized to a great extent to attack the square and multiply, Montgomery ladder and RSA-CRT algorithms.

5 Conclusion

In this paper, we formalize a differential fault analysis on the branch predictor behavior to show that the difference of branch misses for a 2-bit predictor can be utilized to reveal information of the key bits. The attack exploits the strong correlation of the 2-bit dynamic predictor to the unknown underlying branch predictor of the system. Subsequently, we perform validation of our observation in a simulated fault environment on actual computing platforms and show that

the difference of branch mispredictions observed via the performance monitors can be observed by adversaries to determine critical information of secret key bits. Detailed real-life experiments have been performed on several flavors of 1024-bit RSA programs running on Intel platforms where flipping bits in memory is indeed a practical threat. The attacks can be similarly adapted to embedded soft-core processors with practical faults being introduced by instruction skips. Interestingly, fault attack countermeasures which stop or randomize the output when a fault occurs can still be attacked using these techniques. The work raises the question of secured implementation of ciphers in the presence of HPCs in modern processors where fault inductions are feasible.

References

1. Aciiçmez, O., Koç, Ç.K., Seifert, J.-P.: On the power of simple branch prediction analysis. IACR Cryptol. ePr. Arch. **2006**, 351 (2006)
2. Aciiçmez, O., Koç, Ç.K., Seifert, J.-P.: Predicting secret keys via branch prediction. In: Abe, M. (ed.) CT-RSA, Volume 4377 of Lecture Notes in Computer Science, pp. 225–242. Springer, Berlin (2007)
3. Aciiçmez, O., Gueron, S., Seifert, J-P.: New branch prediction vulnerabilities in openssl and necessary software countermeasures. In: Cryptography and Coding, 11th IMA International Conference 2007, Proceedings, pp. 185–203. (2007)
4. AVR Freaks. Instruction skipping after spurious interrupt. http://www.avrfreaks.net/forum/solved-instruction-skipping-after-spurious-interrupt (2015)
5. Bhattacharya, S., Mukhopadhyay, D.: Curious case of rowhammer: flipping secret exponent bits using timing analysis. In: Gierlichs, B., Poschmann, A. (eds.) Cryptographic hardware and embedded systems – CHES 2016. CHES 2016. Lecture notes in Computer Science, vol. 9813, pp. 602–624. Springer, Berlin, Heidelberg (2016)
6. Bhattacharya, S., Mukhopadhyay, D.: Who watches the watchmen?: utilizing performance monitors for compromising keys of RSA on Intel platforms. In: Güneysu, T., Handschuh, H. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2015. CHES 2015. Lecture Notes in Computer Science, vol. 9293, pp. 248–266. Springer, Berlin, Heidelberg (2015)
7. Bhattacharya, S., Rebeiro, C., Mukhopadhyay, D.: Hardware prefetchers leak: a revisit of svf for cache-timing attacks. In: MICRO Workshops, pp. 17–23. IEEE Computer Society (2012)
8. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (eds.) Advances in Cryptology – EUROCRYPT '97. EUROCRYPT 1997. Lecture notes in Computer Science, vol. 1233, pp. 37–51. Springer, Berlin, Heidelberg (1997)
9. Fog, A.: The microarchitecture of Intel, AMD and VIA CPUs/An optimization guide for assembly programmers and compiler makers (2012)
10. Joye, M., Yen, S.-M.: The montgomery powering ladder. In: Kaliski, B.S., Koç, Ç.K., Paar, C. (eds.) CHES, Volume 2523 of Lecture Notes in Computer Science, pp. 291–302. Springer, Berlin (2002)
11. Kim, C.H., Quisquater, J-J.: Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, JJ. (eds.) Information security theory and practices. Smart cards, mobile and ubiquitous computing systems. WISTP 2007. Lecture notes in Computer Science, vol. 4462, pp. 215–228. Springer, Berlin, Heidelberg (2007)
12. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J-H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors. In: ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, 14–18 June 2014, pp. 361–372. IEEE Computer Society, Minneapolis (2014)
13. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: CRYPTO '96 Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, pp. 104–113. Springer, London (1996)
14. Maitra, S., Sarkar, S.: On deterministic polynomial-time equivalence of computing the CRT-RSA secret keys and factoring. IACR Cryptol. ePr. Arch. **2009**, 62 (2009)
15. mp-fpga. Performance counter for microblaze. http://mp-fpga.blogspot.in/2007/10/performance-counter-for-microblaze.html (2007)
16. Patranabis, S., Chakraborty, A., Mukhopadhyay, D.: Fault tolerant infective countermeasure for AES. In: Security, Privacy, and Applied Cryptography Engineering–5th International Conference, SPACE 2015, Jaipur, India, October 3–7, 2015, pp. 190–209 (2015)
17. Rebeiro, C., Mukhopadhyay, D.: A formal analysis of prefetching in profiled cache-timing attacks on block ciphers. IACR Cryptol. ePr. Arch. **2015**, 1191 (2015)
18. Seaborn, M., Dullien, T.: Exploiting the DRAM rowhammer bug to gain kernel privileges. http://googleprojectzero.blogspot.in/2015/03/exploiting-dram-rowhammer-bug-to-gain.html (2015)
19. Seaborn, M., Dullien, T.: Test DRAM for bit flips caused by the rowhammer problem. https://github.com/google/rowhammer-test, 2015 (2015)
20. Weaver, V.M., University of Maine.: Linux perf_event features and overhead. In: 2013 FastPath Workshop (2013)
21. Wikipedia. Rowhammer wikipedia page. https://en.wikipedia.org/wiki/Row-hammer (2016)