Fault Attack revealing Secret Keys of Exponentiation Algorithms from Branch Prediction Misses

Sarani Bhattacharya and Debdeep Mukhopadhyay

***Abstract*—Performance monitors are provided in modern day computers for observing various features of the underlying micro-architectures. However the combination of underlying micro-architectural features and performance counters lead to side-channels which can be exploited for attacking cipher implementations. In this paper, to the best of our knowledge we study for the first time, the combination of branch-predictor algorithms and performance counters to demonstrate a fault attack on the popular square-and-multiply based exponentiation algorithm, used in RSA. The attacks exploiting branching event like branch taken can be foiled by Montgomery Ladder based implementation of the exponentiation algorithm, while attacks based on branch miss are more devastating. We demonstrate the power of the attack exploiting branch misses from performance monitors by formalizing a fault attack model, where the adversary is capable of performing a bit flip at a desired bit position of the secret exponent. The paper characterizes the branch predictors using the popular two-bit predictor and formulates the dependence on the number of branch misses on the fault induced. This characterization is exploited to develop an iterative attack algorithm where knowledge of the previously determined key-bits and the difference of branch misses (as gathered from the performance counters) are utilized to determine the next bit. The attack has been validated on several standard Intel platforms, and puts to threat several implementations of exponentiation algorithms ranging from standard square-and-multiply, Montgomery Ladder to RSA-CRT and which are often used as side-channel counter measures. The attacks show that using the fault attack targeting branch predictors one can attack implementations of exponentiation: both square and multiply, and Montgomery ladder, which forms the central algorithm for several standard public key ciphers.**

***Keywords*-Fault attacks, branch misses, performance counters, Branch Prediction Unit, Square and Multiply Algorithm, Montgomery Ladder Algorithm, RSA-CRT.**

I. Introduction

Micro-architectural features leave footprints in the processor which is often captured by side channels. Side-channel attacks have emerged as a powerful threat to modern cryptographic systems. These attacks allow malicious users to gain access to sensitive data by monitoring **power consumption, timing, or electro-magnetic radiation** of the microprocessor. In modern microprocessors there are several new sources of side channels. The performance improvement measures which are being introduced in recent microprocessors do leak a significant amount of information. For example hardware prefetchers for cache memories are shown to increase leakage in cache-timing attacks [2]. **Cache-timing** attacks monitor the time taken to perform the entire encryption, and relies on statistical means to determine the secret key.

In the pioneering work in [10] it was first shown that the time to process different inputs can be used as a side channel information to find the exponent bits of the secret keys for RSA, Diffie- Hellman, DSS etc. In [1] the penalty for mispredicted branches in number of clock cycles is observed as side channel to identify the data dependent operations of the public key cryptosystem. On a standard RSA implementation, four different types of attacks were performed exploiting the Branch Prediction Unit (BPU) by using both synchronous and asynchronous techniques. Using timing as the side channel in [1], the misprediction information is modeled to identify the secret key. while in the synchronous and asynchronous attacks the Branch Target Buffer (BTB) is modified by the attacker to surface the attack.

Hardware performance counters (HPCs) are a set of special-purpose registers to store the counts of hardware-related activities within the microprocessor. These counters contain rich source of information of the internal activities of the processor and hence can find usage for both attacks and their countermeasures. In [13], these HPCs are exploited as side channels for time based cache attacks. The paper shows that amount of samples required for the attack have a great impact on the different cache eviction strategies. On the other hand, in [5] data from performance counters are used to develop a malware detector in hardware using machine learning techniques. While in [14], a new Virtual Machine Monitor (VMM) named NumChecker is proposed, which exploits HPCs to detect kernel root-kits in a guest Virtual Machine.

In this paper we evaluate the security of cryptographic operations due to the branch events such as **branch misses** observed from the performance counters. The branch information is obtained using **Perf profiling tool** for the modular exponentiation operation performed using both square and multiply and Montgomery ladder algorithms. **Perf** is statistical monitoring tool capable of profiling various events such as branch prediction, branch misses, instruction and data cache hits and misses, cpu cycles etc.

The motivation of this paper is to identify the threats behind the benign information getting revealed while profiling the performance counters. Performance counters can be used to monitor the event **branch-misses** which arise due to branch predictors present in the architecture. We show that if the

The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, 721302, India, E-mail: {sarani.bhattacharya,debdeep}@cse.iitkgp.ernet.in

attacker is able to model the underlying predictor, one can exploit it to determine the secret keys. In this work, we have studied the behavior of the dynamic 2-bit branch predictor as appears in [7] and assumed that the branch mispredictions of a system can be modeled using this 2-bit predictor. From the 2-bit predictor, we can observe the mispredictions for a secret key, but this single piece of information is not enough to identify the individual key bits for a secret key.

In order to have a unique classification of the key-space, we introduce one bit fault at the target bit position of secret exponent and observe the mispredictions for this faulty key using the 2-bit predictor algorithm. This time on the basis of two observations of the branch misses for the secret key and its faulty counterpart, we provide a detailed analysis how the difference of branch misses can be characterized and classified to identify the subsequent key sequence patterns. On the basis of this classification of difference of branch misses from 2-bit predictor, we devise an iterative algorithm which identifies the correct exponent key subsequently one bit after another. Finally, we perform the proposed classification by statistically analyzing the branch-misses from the hardware performance counters and successfully modeled a distinguisher which is capable of revealing secret exponent bits. This one bit fault attack model demonstrates that by using branch misprediction information, and with an assumption of underlying dynamic 2-bit predictor, we can correctly identify the secret key bits. The results demonstrate theoretically, along with validations on several Intel platforms, that by using the information of difference of branch misses from hardware performance counters we are able to retrieve the secret bits uniquely. Thus this paper shows that the performance counter values which are easily accessible to the users at the user privilege can be a potent threat.

The organization of the paper is as follows:- The following Section II provides preliminaries on modular exponentiation algorithms, 2-bit predictor algorithm, fault attack and their countermeasures. In Section III we demonstrate the vulnerability due to the event "branch-misses" as side channels and the adversary fault model. The attack algorithm is described in Section IV with the detailed analysis on the retrieval of secret key bits and the reduction in key space. Classification of the data from hardware performance counters is detailed in Section V. Section VII provides the experimental validations for the attack strategy and the final section contains conclusion of the work we present here.

II. Preliminaries

In this section we provide a brief introduction to the modular exponentiation algorithms and 2-bit dynamic branch predictor algorithm, timing attacks exploiting branch prediction, fault attacks on RSA-CRT and their countermeasures.

A. Modular Exponentiation

In public key cryptography, messages(m) that are encrypted with the public key e perform modular exponentiation as $c = m^e(modn)$ and can only be decrypted as $m = c^d(modn)$ using the private key d in reasonable time. The public key algorithms like RSA and ECC use modular exponentiation during encryption or decryption. The commonly used exponentiation algorithms to implement encryption and decryption for the public key cryptographic algorithms is the square and multiply algorithm. The square and multiply algorithm is described in Algorithm 1. The algorithm performs squaring at each step, while multiplication operation is performed only if the exponent bits are set.

Algorithm 1: Square and Multiply Algorithm

```
Input: y, x = (x_{w-1}, x_{w-2}, ..., 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
```

But this algorithm performs unbalanced instruction execution because multiplication statements are conditioned on the exponent bit. Common side channels such as power, timing are capable of identifying this conditional executions. Simple power attacks (SPA) and timing attacks exploit this conditional instruction execution retrieving the secret exponent.

This paper evaluates the security of implementations of RSA-like ciphers on standard processors. This paper considers a combination of fault attacks and uses the side channels of branch misses leaked from the Performance counters. The branch misses are caused due to branch predictors present in the modern machines. One of the very popular branch predictor algorithm is explained in the next subsection.

B. 2-bit Branch Predictor

This is one of the various predictor algorithms that is most oftenly used in practice. The 2-bit dynamic branch predictor state machine is a deterministic algorithm which predicts next branch to be taken or not taken depending on the history of the branches being taken. Based on the history, if the event predicted by the predictor is not same as the event actually occurring, then it results in a **branch miss**. In a 2-bit prediction scheme the predictor must miss twice before the predicted output changes. The state machine of the dynamic predictor is shown in Figure 1. The state machine has four states, the states either predict the branches to be **taken** or **not taken** depending on the **history of taken branches**. The 2-bit dynamic branch predictor as in Figure 1 has four states S_0, S_1, S_2, S_3, each having a predicted output. On a input, the state transitions are denoted with the arrows, and the labels denote the input bit. The predicted output corresponding to each state is noted in the diagram as state S_0, S_1 predicts 0 and state S_2, S_3 predicts 1.

The branch mispredictions due to the underlying predictor algorithm can be observed by using timing as side channel. The attacks exploiting this side channel leakages are explained in the next subsection.

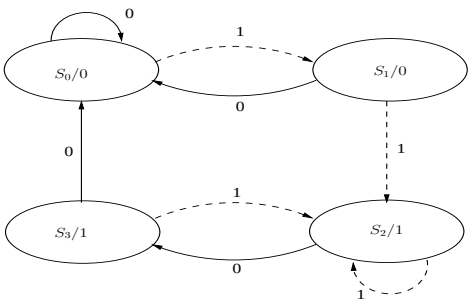

Fig. 1. Dynamic 2-bit Predictor State Machine

C. Timing Attacks on Exponentiation Algorithms using branch prediction

Timing Attacks on Branch Predictors exploit the conditional statements execution in the square and multiply algorithm as in Algorithm 1. The multiplication operation is conditioned on key bits. So if the predicted output from the predictor does not match the observed key bit, a new instruction has to be fetched with a mispenalty. This mispenalty is observed from timing as the side channel source. Thus if the observed timing is more, there is assumed to be a mispenalty. There are various attacks which exploit the branch predictor to correctly model the mispenalties [1]. The countermeasures for these forms of attacks are explained later in subsection II-E1

This paper considers a new kind of fault attacks which determines the key from the difference in branch misses. Here we provide an overview on classical fault analysis of RSA.

D. Fault Attacks on RSA Algorithms

Fault attacks on RSA [3] exploit the vulnerability of the difference of a correct and an incorrect computation caused due to unexpected environmental conditions to retrieve the secret of cryptographic algorithms. Various attacks on public key cipher RSA has been reported in literature which exploit the output of a faulty computation to factor the Modulus N. The difference of correct signature and the faulty signature leaks one of the two large primes in modulus N. A faster efficient implementation on RSA is by using Chinese-Remainder Theorem (CRT) named as the RSA-CRT algorithm which makes decryption four times faster is explained as follows:

Let p and q be two large primes and $N = p \cdot q$ be the RSA modulus. The public encryption key for exponentiation is denoted as e and the private decryption key d is chosen such that $e \cdot d = 1 \ mod(p-1)(q-1)$. To reduce the expense of computation, CRT exponents are calculated as $d_p = d \ mod \ (p-1)$ and $d_q = d \ mod \ (q-1)$. Signature S of a message m is computed as

1) $S_p = m^{d_p} \ mod \ p$
   $S_q = m^{d_q} \ mod \ q$
2) $S = CRT(S_p, S_q) = S_q + q \cdot ((S_p - S_q) \cdot q^{-1} \ mod \ p)$.

In [3] a fault attack is demonstrated on RSA-CRT implementations which can easily factorize the modulus N. The idea for the attack can be represented by the following steps:

- $S_p = m^{d_p} \ mod \ p$, $S_q = m^{d_q} \ mod \ q$ and $S = CRT(S_p, S_q)$
- A fault is introduced while computing S_p or S_q but not both.
- If $\tilde{S}_p$ is the faulty exponentiation, $\tilde{S} = CRT(\tilde{S}_p, S_q) = S_q + q \cdot ((\tilde{S}_p - S_q) \cdot q^{-1} \ mod \ p)$.
- Thus if we take the difference of the correct and the faulty computation, we get $S - \tilde{S} = q \cdot ((S_p - \tilde{S}_p) \cdot q^{-1} \ mod \ p)$
- Secret prime $q = GCD(S - \tilde{S}, N)$ is thus revealed.

The countermeasures for timing and the fault attacks are provided in the next subsection.

E. Countermeasures

1) Countermeasures for Timing Attacks on Exponentiation Algorithm: A naïve modification to protect the side channel leakages of square and multiply exponentiation algorithm is proposed in the Montgomery ladder algorithm [8] as explained in Algorithm 2. This is a well known strategy for protection against simple side channels like timing attacks.

Algorithm 2: Montgomery Ladder Algorithm

```
Input: y, x = (x_{w-1}, x_{w-2}, ..., 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
```

This algorithm performs the entire exponentiation by alternatively modifying the values of two dummy variables depending on the exponent bits. Algorithm 2 has both "if" and "else" statements, and everytime one of the two possible branches are getting executed. Unlike the square and multiply algorithm, here number of branches taken will always be constant and equal to the length of the key. The algorithm executes same number of instructions independent of the exponent, which inhibits simple power attacks and timing attacks.

2) Fault Attack Countermeasures: There are several possible countermeasures to prevent the fault attacks on RSA-CRT implementations proposed by Shamir, Ciet and Joye and Giraud [9]. All of these countermeasures have similar algorithms to prevent the fault attacks. The algorithms have a fault detection logic which check that whether by applying reverse mapping on the observed encrypted output we get back the correct output. The steps to prevent fault attack are:

- Step 1: Computation of two exponentiation
  - Compute $\tilde{S}_p$ and $\tilde{S}_q$
- Step 2: CRT combination
  - Compute $\tilde{S} \leftarrow CRT(\tilde{S}_p, \tilde{S}_q)$
- Fault detection

–

$$Return \begin{cases} S \leftarrow f(\tilde{S}) & \text{if there is no error} \\ \perp & \text{otherwise} \end{cases}$$

In these countermeasures whenever a fault is detected, the algorithm outputs a random value instead of the correct exponentiation result. Thus these countermeasures against standard fault attacks are able to protect the vulnerability of systems featuring the difference of a correct and its faulty computation.

III. Performance Counters As Side Channel Sources

A. Using Branch-misses as Side-channel

In this work, hardware performance counters are exploited to provide side channel information by the **number of branch misses** on the **square and multiply** algorithm, using performance monitoring tools which provide a simple user interface to different event counts. As observed in Algorithm 1, the code execution has two branch paths. Thus, there exists a side channel information via the event **"branch-misses"**. This side channel leakage is caused due to the presence of underlying branch predictors in architecture. Branch misses rely on the ability of the branch predictor to correctly predict future branches to be taken. The branch predictors follow a deterministic algorithm to speculate whether the next branch instruction is to be fetched. If the prediction is false, the instruction pipeline is flushed leading to a branch-miss. Thus the branch predictors play a major role in correctly predicting the next target instruction and reducing mispenalty.

On an event of branch miss, the instruction pipeline flushes and a new target instruction is fetched. So in both square and multiply and Montgomery ladder algorithms, the instruction pipeline undergoes a mispenalty on the event branch miss whenever there is a mismatch in the predicted and observed key bit. This is because the instruction that is going to be executed depends on the key bits. Thus the event branch miss is a strong side channel information which exploits the secret key dependence of the conditional instructions in both the square and multiply and Montgomery ladder exponentiation algorithms.

On the other hand, the fault attack countermeasures also rely on the detection of the fault by a comparison of the correct and incorrect signature. The countermeasure checks for the correctness of the output and produces a random value if it detects a fault. In the meanwhile, number of branch misses for the faulty key sequence can be exploited by an adversary. The performance counters leak benign information of branch misses while exponentiation operation of the faulty computation is performed. The vulnerability of information from the performance counters in presence of a bit flip fault model is modeled in the later section to reveal secret key bits using the properties of the predictor algorithm.

State machine of the 2-bit dynamic predictor as explained in Section II has been extensively used as an underlying predictor in Intel family of microprocessors [6]. In order to validate the behavior, we first installed Perf tool on Linux OS Ubuntu 12.04.1 LTS to monitor the event "branch-misses", which indicates number of branch mispredictions suffered by an executable. The following **command can be executed at the user privilege**.

$ perf stat -e branch-misses executable-name

We first performed a simple experiment by simulating the 2-bit predictor algorithm on a keystream 11 bit exponent in software on Intel i5 platform for performing exponentiation.

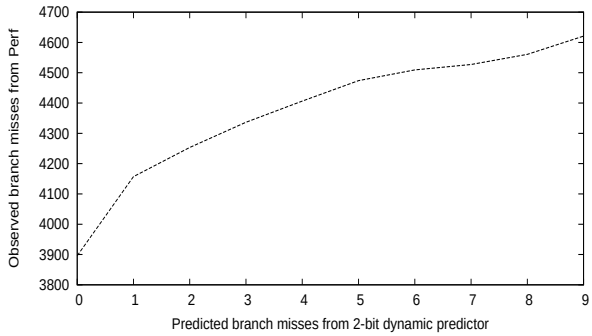

Fig. 2. Variation of branch-misses from performance counters with increase in branch miss from 2-bit predictor algorithm

An observation on the number of branch misses from the predictor and their corresponding performance counter values is illustrated in Figure 2. The number of branch misses obtained from performance counters is found to be increasing as the total number of predicted branch misses on a key-stream increases. Thus, a direct correlation is observed on branch misses from performance counters and branch misses from 2-bit predictor. This confirms our assumption of an underlying 2-bit dynamic predictor having a strong effect on the observed branch misses from the performance counters.

In the next subsection we introduce the concept of transient faults on secret exponent, so that this leads to a exponentiation operation on a transient faulty exponent. The next subsection provides a brief description of the used fault attack model.

B. Adversary Fault Model

In our fault model, we assume that the adversary is capable of introducing a fault in the key bits. 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 execution proceeds with the original secret key. If the adversary is capable of introducing a fault at the i^{th} bit position of the key, then the i^{th} bit of the key gets complemented. The adversary is assumed to follow the **"bit flip model"**. This means that if a key bit at i^{th} position is 1 then it gets flipped to 0 and likewise if 0 then it becomes 1.

1) Fault Algorithm: The adversary is assumed to observe the differences of branch miss between the secret key and its faulty counterpart to efficiently identify the key bits in the following manner:

- Initially the adversary observes the number of branch misses for exponentiation operation using the secret exponent.
- Secondly, he injects the fault at the target bit of secret key, simultaneously observing the number of branch misses for exponentiation using the faulty exponent.

- The information of differences of branch misses of exponentiation operation using exponentiation algorithms is vulnerable to reveal the target bit of secret key.

This information leads to a classification of the key space into several classes based on the observed difference and eventually reveals the key sequence patterns for the respective differences. The key sequences are combinatorially derived with a detailed analysis provided in the next section.

IV. Effect on branch misses due to faulty computation

In this section we show the classification of the key space of key length n bits on the basis of the following assumptions:

1) The length of the key n bit 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) The Adversary introduces a fault at the i^{th} position of the key sequence which eventually flips the i^{th} bit of the key.
4) The Adversary computes the difference in branch misses(Δ_i) for the secret key(K) and the faulty key(F_i).

In the following analysis we will provide a detailed key sequence characterization using the difference of branch misses from the secret and the faulty key. Various parameters used during the analysis are defined as follows: The secret key is denoted by K and the faulty key as F_i each having n bits such that there is one bit fault at the i^{th} position of faulty key. The state of the 2 bit predictor after recovery of j bits for the secret key is St_j^K and for the faulty key is $St_j^{F_i}$. The 2-bit predictor predicts for the $(j+1)^{th}$ bit on the basis of j recovered bits for the secret key as P_{j+1}^K and for the faulty key as $P_{j+1}^{F_i}$.

The number of branch misses for the secret key and the faulty key may or may not be same. The difference in branch misses(Δ_i) between the correct key K and faulty key F_i using a 2-bit predictor algorithm can be atleast -3 and atmost 3. The difference for any key sequence will take any value in the range $[-3, 3]$ and this leads to a partitioning of the key space. We elaborate the above statement with an analysis on maximum difference in branch misses between secret and the faulty key. But before that we state some properties of the predictor states and their relation with the misprediction.

It may be emphasized that the secret key $(b_0, b_1, \cdots, b_{n-1})$ and the faulty key differs only in the i^{th} bit. We state 4 properties using the state transition of the 2 bit dynamic predictor.

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 i^{th} bit. Also from Figure 1, in states S_0 & S_2, the last input bit i.e, the $(i-1)^{th}$ bit is same as the i^{th} predicted bit.

Property 2: If $St_{i-1}^K = S_0$ or $St_{i-1}^K = S_2$, then there is guaranteed mispredictions at the i^{th} bit for either K or F_i. If the $(i+1)^{th}$ bit is not same as the predicted P_i^K, then there is a mismatch in the predictor's output for the $(i+2)^{th}$ position $P_{i+2}^K \neq P_{i+2}^{F_i}$.

- Either b_i or $\overline{b_i}$ results in a misprediction for the i^{th} bit of the secret or the faulty key. Since $b_{i-1} = P_i^K$, atleast two misprediction are required at $i, (i+1)^{th}$ position to make $P_{i+2}^K \neq P_{i+2}^{F_i}$. Thus if $b_{i+1} \neq P_{i+1}^K$, then there will be two consecutive mispredictions at $i, (i+1)^{th}$ position either for the secret or the faulty key resulting in $P_{i+2}^K \neq P_{i+2}^{F_i}$.

Property 3: If $St_{i-1}^K = S_1$ or $St_{i-1}^K = S_3$ then $b_{i-1} \neq P_i^K = P_i^{F_i}$

- Since $(i-1)$ bits are identical, $P_i^K = P_i^F$ and from Figure 1 it clearly follows that the last input bit to the predictor algorithm does not match the predicted output if the state is S_1/S_3.

Property 4: If $St_{i-1}^K = S_1$ or $St_{i-1}^K = S_3$, a single misprediction at i^{th} bit is sufficient to make $P_{i+1}^K \neq P_{i+1}^{F_i}$.

- Since $b_{i-1} \neq P_i^K = P_i^F$, both the secret and faulty key suffers from a misprediction at b_{i-1}. At the i^{th} position, either b_i or $\overline{b_i}$ results in a misprediction. So either for the secret key or the faulty key, there are two consecutive mispredictions at the $(i-1), i^{th}$ bits. Thus the predicted output for the $(i+1)^{th}$ bit changes. So a single guaranteed misprediction at the $(i)^{th}$ bit changes results in $P_{i+1}^K \neq P_{i+1}^{F_i}$.

Without loss of generality, we aim to prove the bounds for the difference in branch misses by maximizing the branch misses from the faulty key and minimizing the branch miss from the secret key.

In order to maximize the difference of branch misses, effectively we minimize the branch misses for the secret key as well as maximize the branch misses for the faulty key. In the following discussion we are going to analyse the difference of branch misses for two different scenarios.

a) **Case 1**: If $St_{i-1}^K = S_1$ or $St_{i-1}^K = S_3$
Before going into the formal analysis, we explain this with a simple example. Let us assume a small secret key as 1001000100 and a fault is introduced at 5^{th} position from the MSB, thus the faulty key is 1001**1**00100. The 5^{th} position of faulty key (from the left) differs from the secret key and lets assume that we already know 4 bits as 1001. After the initial 4 bits the predictor state is at S_1, and predicts a 0.

- As the immediate next bit i.e, the 5^{th} is 0 for the secret key there is no change in branch miss, but for the faulty key there is a bit flip at the 5^{th} position. So the faulty key suffers from a branch miss increasing the difference by 1.
- At this stage, the predicted value for secret key is 0 but for the faulty key it is 1 as it suffered from two mispredictions from two consecutive 1's.
- So if the next two bits are again zero then for secret key there is no increase in branch miss.

- But for faulty key, there are two mispredictions due to the mismatch of the predicted value as 1 and the bit values at $6, 7^{th}$ position as 0.
- Thus the difference of branch misses between the secret and the faulty key is 3.
- This is the maximum difference because, after getting two consecutive zeros at the $6, 7^{th}$ position the predictor's output for the secret and the faulty key becomes same. Thus further increase in branch misses gets canceled as they occur in both secret and its faulty counterpart and do not add up to the difference.

Now, we formalize the above example with a generalized structure using the Figure 3.

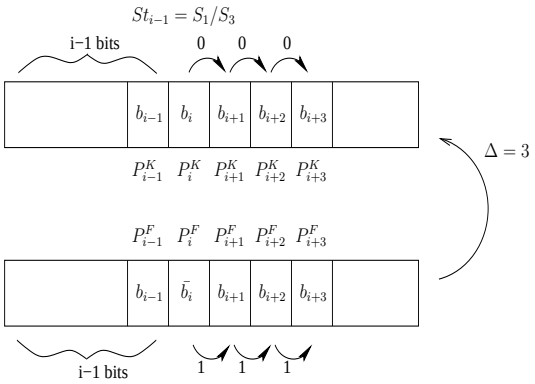

Fig. 3. Maximum difference in branch misses between the secret and i^{th} faulty key

- $P^K_i = P^{F_i}_i$. Every predicted bits from 2-bit branch predictor for 0^{th} to i^{th} bit are same for both keys.
- By Property 3, $b_{i-1} \neq P^K_i$ the last input bit is not same as the predicted value.
- **Case 1(a)**: Minimum number of branch misses for the secret key,
  - This occurs when $b_i = P^K_i$, current bit matches with the predicted bit of the secret key.
  - Further, as in Figure 1, after the state transition of i bits the states move to S_0/S_2, having the same predicted output bit $P^K_{i+1} = P^K_i$.
- **Case 1(b)**: On the contrary for the faulty key
  - $\overline{b_i} \neq P^K_i = P^F_i \Rightarrow \overline{b_i} \neq P^F_i$, thus branch miss increases by 1.
  - and by property 4 due to two mispredictions at $(i-1), i^{th}$ positions of the faulty key, the output of the predictor for the secret and faulty key differs as $\Rightarrow P^F_{i+1} \neq P^F_i = P^K_i = P^K_{i+1}$. This is due to $b_{i-1} = \overline{b_i} \neq P^F_i$ and predicted bit of $(i+1)^{th}$ position differs.

  Thus after i bits the differences in branch miss is 1 and predicted output $P^K_{i+1} \neq P^F_{i+1}$ resulting in mismatch.
- **Case 1(c)**: To minimize the number of misses of secret key, if b_{i+1}, b_{i+2} is chosen such that, $b_{i+1} = b_{i+2} = P^K_i$, there will be no increase in branch miss of secret key. The predicted outputs also remain the same as $P^K_{i+1} = P^K_i$, $P^K_{i+2} = P^K_{i+1}$ and $P^K_{i+3} = P^K_{i+2}$.
- **Case 1(d)**: While for the faulty key, $b_{i+1} = b_{i+2} \neq P^F_{i+1}$ as we already stated that $P^F_{i+1} \neq P^K_{i+1}$, this results in two mispredictions adding up the difference in branch misses to 3.
- **Case 1(e)**: For the $(i+3)^{rd}$ bit,
  - in the faulty key, due to two consecutive mispredictions at $(i+1), (i+2)^{th}$ position the predicted bit for faulty key flips as $P^F_{i+3} \neq P^F_{i+1}$. Thus $P^F_{i+3} = P^F_i = P^K_i = P^K_{i+3}$.
  - Both for the secret key and the faulty key, $P^K_{i+3} = P^F_{i+3}$, and the next bits are identical for both the keys.
- So the further increase in branch misses being same in both keys does not add up to the difference.

Thus the maximum difference of branch misses between the secret key and i^{th} bit faulty key sequence can be atmost 3 when $(i-1)^{th}$ bit is either at S_1 or S_3.
Next we analyze the maximum difference in branch misses in the scenario where $St^K_{i-1} = S_0$ or S_2.

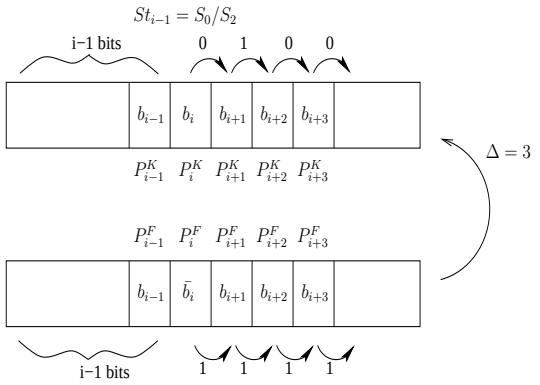

Fig. 4. Maximum difference in branch misses between the secret and i^{th} bit faulty key

b) **Case 2**: If $St^K_{i-1} = S_0$ or $St^K_{i-1} = S_2$
When the predictor state St^K_{i-1} is at S_0 or S_2 the analysis is similar to the previous one, but here the mismatch of predicted output of the faulty and the secret key is observed at the $(i+2)^{th}$ bit instead of $(i+1)^{th}$ bit. The mismatch of predicted bit of the partial secret and the faulty key is essential to maximize the difference, because otherwise the difference would not be exceeding 1. Thus to maximize the difference, by Property 2 there must be two subsequent mispredictions to change the predicted output if $St^K_{i-1} = S_0$ or S_2. The formal analysis is provided as follows along with the Figure 4:

- $P^K_i = P^F_i$, since every predicted bits from 2-bit branch predictor for 0^{th} to i^{th} bit are same for both keys.
- By Property 1, since the state is either S_0/S_2 the $(i-1)^{th}$ input bit is same as the predicted value for the i^{th} bit, $b_{i-1} = P^K_i = P^F_i$.
- **Case 2(a)**: Choosing the i^{th} bit such that there is minimum number of branch misses for the secret key,
  - To minimize the number of misses for the secret key, i^{th} bit is chosen same as the predicted bit $b_i = P^K_i$,

and branch miss does not increase.
  - Again for the secret key, predictor on getting bit $b_i = P_i^K$, predicts $P_{i+1}^K = P_i^K = b_i$ same as the previous predicted value.
- **Case 2(b)**: On the contrary, for i^{th} bit of the faulty key
  - $\overline{b_i} \neq P_i^F$, and branch miss increases by 1.
  - Again since $b_{i-1} \neq \overline{b_i}$ and $\overline{b_i} \neq P_i^{F_i}$, there is a single misprediction for the i^{th} bit. A single misprediction is not enough to cause the mismatch in the predicted output of the secret and the faulty key bits. Thus the predicted output for the $i+1^{th}$ bit for the faulty key does not change as $P_{i+1}^F = P_i^F = b_i$.
- **Case 2(c)**: In order to maximize the difference in branch misses, $(i+1)^{th}$ bit for secret and faulty key is chosen to be $\overline{b_i}$ because,
  - if $b_{i+1} = \overline{b_i}$, then branch miss for both keys 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, ($\overline{b_i} = b_{i+1} \neq P_{i+1}^F = P_{i+1}^K$,) prediction bit for the secret and the faulty key for the $(i+2)^{th}$ bit differs as $P_{i+2}^F \neq P_{i+1}^F = P_{i+2}^K$.
- **Case 2(d)**: Again, to minimize the number of misses of secret key, we assume $b_{i+2} = b_{i+3} = P_{i+1}^K = P_i^K$, so there is no increase in branch misses for secret key.
- **Case 2(e)**: While $P_{i+2}^F \neq b_{i+2} = b_{i+3}$, increasing the branch miss of faulty key by 2.
- **Case 2(f)**: For the $(i+4)^{th}$ bit for secret and faulty key,
  - Finally, for the $(i+4)^{th}$ bit the predicted value for faulty key flips, $P_{i+4}^F \neq P_{i+2}^F$
  - Thus for both for the secret key and the faulty key, after $i+4$ bits the predicted value becomes equal as $P_{i+4}^K = P_{i+4}^F$, the next bits being identical for both the keys, has no effect in changing the difference value.

Thus maximum difference of branch misses between the secret key and i^{th} bit faulty key sequence can be atmost 3 when $(i-1)^{th}$ bit is either at S_0 or S_2. A similar analysis can be equivalently adapted for all specific states (S_0, S_1, S_2, S_3) and difference values in the range [-3,3].

The extent of correctly retrieving the key bits may vary with the states of the previous key bits as well as the observed Δ_i's. In the next subsection we use the similar analysis in specific scenario like $St_{i-1}^K = S_0$ to retrieve the subsequent key sequence patterns for different values of Δ_i.

A. Reduction in Key Space

As explained in the previous subsection, the difference of branch misses simulated through 2-bit predictor algorithm can belong to any class from $[-3, 3]$ for any key sequence of arbitrary length. Similar analysis of differences in branch miss leads to a classification for each value in the range $[-3, 3]$. Using this classification, we devise an adversary algorithm, which retrieves consecutive key bits one after another starting from the Most Significant Bit (always 1). Thus at any point of time, while determining any intermediate i^{th} secret bit, assumption follows that

- Previous $(i-1)$ bits are known. This enables the adversary to precalculate the state of the predictor for $(i-1)$ bits.
- Given the state St_{i-1} after $(i-1)$ bits, the difference(Δ_i) of branch misses from secret and i^{th} bit faulty key for exponentiation operation, leads to a unique identification of the i^{th} bit as well as the subsequent bits as shown in Tables I.

TABLE I
KEY SEQUENCE PATTERN OF $b_i, b_{i+1}, \cdots, b_{n-1}$ FOR ONE BIT FAULT

St_{i-1}	Δ_i						
	-3	-2	-1	**0**	**1**	**2**	**3**
S_0	110	$1(10)^*$	10	_1	00	$0(10)^*$	010
S_1	10	1	-	-	-	0	00
S_2	001	$0(01)^*$	01	_0	11	$1(01)^*$	101
S_3	01	0	-	-	-	1	11

In the following subsection we describe the characterization of key sequence patterns by observing the differences of branch misses, when the previous key bits are already known to the adversary.

1) **When $St_{i-1} = S_0$:** The key sequence characterization for $St_{i-1} = S_0$ is explained for differences observed as $\Delta_i = 0, 1, 2, 3$. In the following 4 cases we show the analysis behind the unique classification and identification of subsequent key bits as in Table I.

a) Case 1: When $\Delta_i = 0$
By introducing a single bit fault at the i^{th} position, the difference cannot characterize the i^{th} bit but can successfully do for the $(i+1)^{th}$ bit. The initial analysis is provided for the correct identification of the $(i+1)^{th}$ bit.
- The predictor state machine on encountering $(i-1)$ bits is at state S_0.
- Thus, from Property 1, $b_{i-1} = 0 = P_i^K = P_i^F$.
- The i^{th} bit b_i can either be 0 or 1.
  - If $b_i = 0$, then $b_i = P_i^K$ but $\overline{b_i} \neq P_i^F$ and $\Delta_i = 1$.
  - On the other hand, if $b_i = 1$, then $b_i \neq P_i^K$ but $\overline{b_i} = P_i^F$ and $\Delta_i = -1$.

  Difference of branch misses from one bit fault cannot determine the i^{th} bit uniquely.
- To ensure $\Delta_i = 0$, b_{i+1} is uniquely identified as 1.
  - For the $(i+1)^{th}$ bit $P_{i+1}^K = P_{i+1}^F = P_i^k$.
  - Now, if $b_{i+1} = 0$, $\Rightarrow b_{i+1} = P_{i+1}^K = P_{i+1}^F$, this implies that $P_{i+2}^K = P_{i+2}^F$ and subsequent bits for secret and faulty key being identical, no change in Δ_i. Thus Δ_i for the entire key either becomes 1(when $b_i = 0$), or -1(when $b_i = 1$) and can never be 0. So $b_{i+1} = 0$ is impossible when $\Delta_i = 0$.
  - So if and only if b_{i+1} is certainly 1, state after $(i+1)^{th}$ bit for the secret and the faulty key will change, so as to observe the mismatch in predicted value for

the $i+2^{th}$ bit, $P^K_{i+2} \neq P^F_{i+2}$. This is essential to make the difference $\Delta_i = 0$.

Thus from this analysis, we say if $St_{i-1} = S_0$ and observed difference $\Delta_i = 0$, then the i^{th} bit b_i can either be 0 or 1, but b_{i+1} is uniquely identified as 1.

b) Case 2: $\Delta_i = 1$

The key sequence for this case can be identified uniquely with a single bit fault model as,

- The predictor state machine on encountering $(i-1)$ bits is at state S_0.
- Again from Property 1, $b_{i-1} = 0 = P^K_i = P^F_i$.
- If $b_i = 0$, then $b_i = P^K_i$. Thus there is no branch miss for the secret key, but the faulty key observes $\overline{b_i}$ and experiences a branch miss. Since $\overline{b_i} = 1 \neq P^F_i$ and due to this prediction $\Delta_i = 1$.
- For the $i+1^{th}$ bit, if and only if $b_{i+1} = 0$, $(i+1)^{th}$ predicted bit $b_{i+1} = P^K_{i+1} = P^F_{i+1}$, and the further changes in branch miss would not get reflected in their difference.

Thus we conclude, for $St_{i-1} = S_0$ and $\Delta_i = 1$, then $(b_i, b_{i+1}) = (0, 0)$.

c) Case 3: $\Delta_i = 2$

In order to determine the subsequent bit in this case, we aim to minimize the number of branch miss of the secret key so that the difference in branch miss between the faulty and the secret key is a positive value.

- Since the state after $(i-1)$ bits is S_0, $b_{i-1} = 0$ and $P^K_i = P^F_i = 0$.
- b_i is uniquely identified as 0 as
  - In order to observe a positive Δ_i, b_i must be 0 for which there is misprediction in the faulty key sequence as $\overline{b_i} = 1 \neq P^F_i$, and difference up to i bits is 1.
- b_{i+1} is also uniquely identified to be 1,
  - As explained in the case $\Delta_i = 1$, if $b_{i+1} = 0$ then Δ_i will remain equal to 1. In order to increase Δ_i to 2, b_{i+1} must be equal to 1. For $b_{i+1} = 1$, there is a misprediction both for the secret as well as the faulty key, and thus the difference Δ_i still remains as 1.

    The $i+1^{th}$ bit being 1, for the faulty key there are to 2 subsequent mispredictions at $i, i+1^{th}$ bit, thus $b_{i+1} = 1$ leads to mismatch in prediction value for the $(i+2)^{th}$ bit, $P^F_{i+2} = \overline{P^F_{i+1}} = 1 \neq P^K_{i+2} = 0$.
- $b_{i+2} = 0$ in order to increase Δ_i to 2,
  - Again in order to increase Δ_i, $b_{i+2} = 0$, resulting in no branch miss for secret key as $P^K_{i+2} = 0$. But the faulty key having $P^F_{i+2} = 1$ suffers a misprediction, increasing Δ_i to 2.

  Thus at this point, $P^K_{i+2} = 0$ and $P^F_{i+2} = 1$.
- The pattern follows, like to maintain a difference Δ_i of 2, an alternating sequence of $(1, 0)$ has to be maintained henceforth because, for a pair of $(1, 0)$ - secret key suffers a misprediction for 1, similarly faulty key suffers a misprediction for 0. Thus this alternating sequence has to be maintained to observe $\Delta_i = 2$ for $St^K_{i-1} = S_0$.

  For this case, if $St_{i-1} = S_0$ and $\Delta_i = 2$, then the subsequent key sequence is $0(10)^*$. Thus $(b_i, b_{i+1}, b_{i+2}) = (0, 1, 0)$ and if the number of bits yet to be retrieved $n-(i+2)$ is even, then it is an alternation of $(1, 0)$ bits.

d) Case 4: $\Delta_i = 3$ The analysis in this case is similar to the proof for the maximum difference that we provided in the previous subsection.

- From property 1, state is S_0 after $i-1$ bits, $b_{i-1} = 0 = P^K_i = P^F_i$
- b_i is uniquely determined to be 0,
  - To minimize the number of misses for the secret key, $b_i = 0 = P^K_i$, branch miss does not increase.
  - On the contrary, for the faulty key $P^F_i \neq \overline{b_i}$, branch miss increases by 1.
  - Predictor on getting bit $b_i = P^K_i$, predicts $P^K_{i+1} = P^K_i = b_i = P^F_i = P^F_{i+1}$.
- b_{i+1} has to be 1 in order to make $\Delta_i > 1$,
  - Now, if $b_{i+1} = 1$, then branch miss for both keys increases by 1, and for the two consecutive mispredictions of faulty key following property 2, $\overline{b_i} = b_{i+1} \neq P^F_{i+1} = P^K_{i+1}$, prediction bit for the secret and the faulty key for the $(i+2)^{th}$ bit differs as $P^F_{i+2} = \overline{P^F_{i+1}} = 1 \neq P^K_{i+2}$.
- b_{i+2} is uniquely determined as 0 in order to make $\Delta_i = 2$,
  - Again, if we assume $b_{i+2} = 0 = P^K_{i+1} = P^K_i$, so there is no increase in branch misses for secret key.
  - But $b_{i+2} \neq P^F_{i+2}$, increasing the branch miss of faulty key by 1 making the difference $\Delta_i = 2$.
- b_{i+3} can either be $0/1$,
  - If $b_{i+3} = 0$, then straightaway Δ_i becomes 3 and attains the maximum value.
  - Else if $b_{i+3} = 1$, then Δ_i becomes 1. For this case, again a pattern of $(0, 1)$'s can be observed terminating with consecutive $(0, 0)$'s making the $\Delta_i = 3$.

  Thus b_{i+3} can either be $0/1$ and cannot be uniquely determined when $\Delta_i = 3$.

Thus for $St_{i-1} = S_0$ if and only if $(b_i, b_{i+1}, b_{i+2}) = (0, 1, 0)$ then $\Delta_i = 3$. The number of bits correctly retrieved in this case is the highest.

In the analysis of retrieving the unknown bits, we are able to correctly determine the target bit b_i for some cases but not for all of the cases. As in our previous discussion in Case IV-A1 we were able to determine $(i+1)^{th}$ bit but not the i^{th} bit. In order to correctly identify the i^{th} bit, we introduce 2 bit faults. In the next subsection, we provide the modeling for two bit fault model.

B. Limitations of One bit fault model

By using one bit fault model, we are able to retrieve 20 out of 28 cells in the Table I. Our aim is to correctly identify the i^{th} bit of the secret exponent. So for this we introduce a 2-bit fault model where we observe the difference in branch miss between the $(i+1)^{th}$ bit faulty key F_{i+1} and another

faulty key $F_{i,i+1}$ with a 2 bit fault at positions $i, i+1$. The difference of branch misses as observed from this two faulty keys are denoted as $\Delta_{i,i+1}$. We classify the key sequences for this two bit fault model observed for $St_{i-1} = S_0/S_2$ having $\Delta_i = 0$ in the following Table II.

TABLE II
KEY SEQUENCE PATTERN OF b_i, b_{i+1} FOR TWO BIT FAULT

St_{i-1}^K \ $\Delta_{i,i+1}$	-1	**1**
S_0	11	01
S_2	00	10

In the subsection IV-A1, using a single bit fault model we failed to determine b_i. The following analysis reveals the i^{th} bit with a 2-bit fault.

- In the previous analysis in subsection IV-A1, it was stated that if $b_{i+1} = 0$ instead of 1, then the overall difference of branch misses would have been 1 or -1.
- This information can be exploited to identify b_i.
- If a fault is injected at $i + 1^{th}$ position, b_{i+1} becomes 0 and the number of branch misses are observed.
- Secondly, two bit faults are injected at b_i, b_{i+1} and the difference $\Delta_{i,i+1}$ is observed.
- As analyzed previously, if observed difference
  - $\Delta_{i,i+1} = 1$ then $b_i, b_{i+1} = (0, 1)$,
  - $\Delta_{i,i+1} = -1$ then $b_i, b_{i+1} = (1, 1)$.

The complete characterization for two bit fault model is shown in Table II. Thus, in the next subsection we demonstrate either with a 1 bit or 2 bit fault we are able to perform a complete characterization of next bit b_i as well as b_{i+1}, given a partial key sequence. This form a sequential algorithm which gradually identifies the entire key sequence by correct characterization of the immediate next bits. For each cell for Table I, we can form a similar analysis to retrieve the subsequent key sequences. Some of the cells will retrieve more key bit as the cells for row S_0/S_2, some very less as S_1/S_3. Using this information we are going to devise an adversary attack model which deterministically retrieves key bits sequentially one after another.

C. Adversary Attack Algorithm

The adversary attack model as illustrated with the following Algorithm 3 observes the difference of branch misses from the secret key and its corresponding faulty key. The difference of branch misses can be successfully used to retrieve the unknown key bits. The algorithm starts with an assumption that the length of the secret key is known to the adversary and the most significant bit is always 1. This algorithm gradually retrieves one unknown key bit at a time from the most significant bit (left) to the least significant bit (right). To determine the i^{th} bit value from the left, the adversary already has the knowledge of the previous $i-1$ bits. Following the 2 bit predictor algorithm, $i-1$ transitions over the states S_0, S_1, S_2, S_3 can be traced by the adversary.

The attack is modeled as follows: the adversary starts from the most significant bit (always 1), observes the difference of the branch misses for the unknown secret key and the faulty key. The difference of branch misses for i^{th} bit faulty key and the secret key can be classified uniquely with the prior knowledge of the state of the 2-bit predictor after $i-1$ transitions.

The observations related to the nature of the key bit sequences and their relation with the difference of branch misses are listed below for two different scenarios:

1) Case 1: If $St_{i-1} = S_0/S_2$
   - If the difference of branch misses is $(-3, -2, -1, 1, 2, 3)$, then the i^{th} bit can be uniquely identified with a single bit fault at i^{th} position.
   - As in Table I, if the difference of branch misses is 0, then the i^{th} bit cannot be uniquely determined, but $(i+1)^{th}$ bit can be uniquely identified.
   - In order to identify the i^{th} bit, difference of branch misses are observed for the secret key with $(i+1)^{th}$ bit flipped and with secret key having both $i, (i+1)^{th}$ bits flipped. In this case the difference classes uniquely retrieve the i^{th} bit.

   Thus any key sequence with $St_{i-1} = S_0$ or S_2 can be uniquely retrieved with one bit flip, if not then with 2 bit flips.
2) Case 2: If $St_{i-1} = S_1/S_3$
   - If the difference of branch misses is $(-3, -2, 2, 3)$, then the i^{th} bit can be uniquely identified with a single bit fault at i^{th} position.
   - If the difference of branch misses is $(-1, 0, 1)$, then the i^{th} bit cannot be determined with a single bit fault.

   Thus 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 i^{th} secret bit.

The entire classification is explained below with Algorithm 3 and two Tables I and II

In order to check the correctness of the algorithm, we developed a software simulation of the fault attack setup, where the 2-bit branch prediction state machine is simulated to provide the branch misprediction information. The software simulation of a fault is implemented by complementing the target position of the secret exponent and the difference of branch misprediction information is required to reveal the secret key bits one after another. The software simulation successfully retrieved individual key bits by using the algorithm. The simulation runs as an iterative algorithm retrieving one bit after another.

In the next section, we will demonstrate the classification observing the data from hardware performance counters and its limitations. The data from the performance counters in differences of branch misses follows a pattern. The next section provides a brief introduction as to how data from performance counters are profiled and classified.

Algorithm 3: Attack Algorithm

Input: Unknown key(k) of n bit $(k_0, k_1, \cdots, k_{n-1})$
Output: Retrieved key bits $(k_0, k_1, \cdots, k_{n-1})$
begin
  Assume $k_0 = 1$;
  for $i = 1$ *to* $n-1$ **do**
    Set $Flag = 0$;
    St_{i-1} = state of branch predictor for known $i-1$ bits;
    bm_0 = number of branch misses for secret key;
    bm_1 = branch misses for the secret key with fault at i^{th} bit;
    $\Delta_i = bm_1 - bm_0$;
    if *($St_{i-1} = S_1$ or $St_{i-1} = S_3$)* **then**
      if *($\Delta_i = -1$ or $\Delta_i = 0$ or $\Delta_i = 1$)* **then**
        bm_0 = branch misses with fault at $(i-1)^{th}$ bit;
        bm_1 = branch misses with fault at $(i-1)$, i^{th} bit;
        $\Delta_{i-1,i} = bm_1 - bm_0$;
        if *($\Delta_{i-1,i}$ = 0)* **then**
          Set $Flag = 1$;
        end
        if *($St_{i-1} = S_1$)* **then**
          Set $St_{i-1} = S_0$;
        else
          Set $St_{i-1} = S_2$;
        end
      else
        Access Table I corresponding to S_1 or S_3 at location Δ_i;
      end
    end
    if *($St_{i-1} = S_0$ or $St_{i-1} = S_2$)* **then**
      if *($\Delta_i = 0$)* **then**
        if *($Flag = 1$)* **then**
          bm_0 = branch misses with fault at $(i-1)$,$(i+1)^{th}$ bit;
          bm_1 = branch misses with fault at $(i-1)$,i, $(i+1)^{th}$ bit;
          $\Delta_{i-1,i,i+1} = bm_1 - bm_0$;
        else
          bm_0 = branch misses with fault at $(i+1)^{th}$ bit;
          bm_1 = branch misses with fault at i, $(i+1)^{th}$ bit;
          $\Delta_{i,i+1} = bm_1 - bm_0$;
        end
        if *($\Delta_{i-1,i,i+1}$ = -1 or $\Delta_{i,i+1}$ = -1)* **then**
          Access Table II corresponding to S_0/S_2 at respective difference location;
        end
        if *($\Delta_{i-1,i,i+1}$ = 1 or $\Delta_{i,i+1}$ = 1)* **then**
          Access Table II corresponding to S_1/S_3 at respective difference location;
        end
      else
        Access Table I corresponding to S_0 or S_2 at location Δ_i;
      end
    end
  end
  Update the partial known key to $(k_0, k_1, \cdots, k_i)$.
end

V. Classification with data from hardware performance counters

The algorithm as explained in the previous section is modeled with absolute values of branch misses as obtained by calculating mispredictions from 2-bit predictor algorithm. In this section, we will demonstrate how we profile event "branch misses" obtained through the hardware performance counters for random keys, and then eventually perform a classification using the difference of branch mispredictions such that we can efficiently determine the subsequent key sequences.

In order to perform an efficient classification, we initially perform a profiling with a set of key generated at random. This can be viewed as a **learning phase** for correctly identifying the classes to perform a classification using hardware performance counters. In the second phase, we wish to classify a branch misprediction profile of an observed key to its respective difference class leaking the next key sequences.

1) Learning Phase: In the learning phase, we aim to learn a correlation between the theoretical difference of branch misses from the 2-bit branch predictor and the differences obtained from actual hardware performance counters in real processors.

A set of random keys are generated. These keys are classified on the basis of the difference of branch mispredictions (Δ_i) from 2-bit predictor algorithm (between original key and its faulty counterpart) into respective classes of difference in the range $[-3, 3]$. Thus we have multiple keys and their respective faulty key sequences classified according to observed Δ_i from 2-bit predictor algorithm.

Next by using information of branch mispredictions obtained from hardware performance counters, we construct the distributions of branch misses by varying plaintexts for each of the random keys as well as for their faulty counterparts. Thus for all pair of secret and its fault keys belonging to a particular difference class Δ_i, we perform the following:

- The difference of branch misses are observed from hardware performance counters over a set of 1000 plaintexts and the mean value of the distribution is calculated.
- The mean values from this difference distribution for a set of keys (belonging to a specific Δ_i class) forms another distribution of mean differences (over a set of keys having same Δ_i).

One such distribution of difference of branch mispredictions for all possible Δ_i's (from 2-bit prediction) for a set of random keys of length $n = 21$ bits, averaged over a set of 1000 random plaintexts is showed in Figure 5. The mean values for the distributions are tabulated in the first row of Table III for respective Δ_i's.

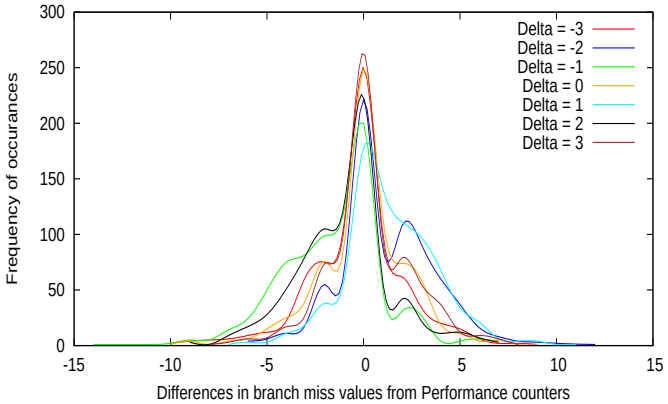

Fig. 5. Distribution of Difference in branch misses between the secret and i^{th} bit faulty key from performance counters on Intel Core i5

As from the Figure 5 it is clear that all the distributions are overlapping, and it is hard to correlate the distributions with

respect to the theoretical differences from branch prediction algorithm(Δ_i). In case of the theoretical branch predictors, this classification into difference classes from -3 to 3 was easier since they were obtained as discrete values. But the classification becomes difficult when the data is observed from actual performance counters on real processors.

To overcome this limitation, in the next section we combine one and two bit faults such that the posed classification problem becomes easier. The theoretical analysis for this combination is described in the next section followed by results using data from hardware performance counters.

VI. Combining One and Two bit fault model

Combination of one and two bit fault retrieves the bit positions in the keystream more efficiently. We will eventually devise an efficient attack algorithm which works successfully using the branch misprediction information from hardware performance counters. The scenarios are not same for all the states in Table I. We initially provide a detailed analysis for $St^K_{i-1} = S_0/S_2$. Let $\Delta_{i,i+1}$ be the difference of branch misses from a 2 bit and 1 bit fault.

2) **When $St^K_{i-1} = S_0/S_2$:** The analysis of combining fault models is explained with the following observations:

a) **If** $St^K_{i-1} = S_0$

The following analysis shows that by combining Tables I and II we can classify bits (b_i, b_{i+1}) on the basis of differences of branch misses from a one bit fault and a two bit fault.

- From Table I, we observe that if $St^K_{i-1} = S_0$, $\Delta_i = 1$, then $(b_i, b_{i+1}) = (0, 0)$.
- For, $\Delta_i = -1$, then $(b_i, b_{i+1}) = (1, 0)$.
- If $\Delta_i = 0$,
  - then $(b_i, b_{i+1}) = (-, 1)$ from table I.
  - To identify b_i, we observed difference between a 1 bit and 2 bit fault as $\Delta_{i,i+1}$.
  - From Table II if $b_i = 0$, then $\Delta_{i,i+1} = 1$, else if $b_i = 1$, then $\Delta_{i,i+1} = -1$.
  - So, if $St^K_{i-1} = S_0$, $\Delta_i = 0$ and $\Delta_{i,i+1} = 1$ then $(b_i, b_{i+1}) = (0, 1)$.
  - Again, if $St^K_{i-1} = S_0$, $\Delta_i = 0$ and $\Delta_{i,i+1} = -1$ then $(b_i, b_{i+1}) = (1, 1)$.
- On the other hand if $\Delta_i = 2/3$, from Table I we observe that $(b_i, b_{i+1}) = (0, 1)$, and in this case $\Delta_{i,i+1}$ is also 1.
- Similarly if $\Delta_i = -2/-3$, from Table I we observe that $(b_i, b_{i+1}) = (1, 1)$, and in this case $\Delta_{i,i+1}$ is -1.

Using this observation, we build a simple classification for consecutive two bits (b_i, b_{i+1}) as below:

- If $(\Delta_i = 1)$, then $(b_i, b_{i+1}) = (0, 0)$.
- If $(\Delta_i = -1)$, then $(b_i, b_{i+1}) = (1, 0)$.
- If $(\Delta_i \neq 1/-1)$ & $(\Delta_{i,i+1} = 1)$, then $(b_i, b_{i+1}) = (0, 1)$.
- If $(\Delta_i \neq 1/-1)$ & $(\Delta_{i,i+1} = -1)$, then $(b_i, b_{i+1}) = (1, 1)$.

b) **If** $St^K_{i-1} = S_2$

This case can be analyzed in the exactly same way as the previous case $St^K_{i-1} = S_0$. The classification for consecutive two bits (b_i, b_{i+1}) for $St^K_{i-1} = S_2$ is as below:

- If $(\Delta_i = 1)$, then $(b_i, b_{i+1}) = (1, 1)$.
- If $(\Delta_i = -1)$, then $(b_i, b_{i+1}) = (0, 1)$.
- If $(\Delta_i \neq 1/-1)$ & $(\Delta_{i,i+1} = 1)$, then $(b_i, b_{i+1}) = (1, 0)$.
- If $(\Delta_i \neq 1/-1)$ & $(\Delta_{i,i+1} = -1)$, then $(b_i, b_{i+1}) = (0, 0)$.

3) **When $St^K_{i-1} = S_1/S_3$:** While modelling the adversary attack Algorithm 3 we observed that, while retrieving the i^{th} bit, if $St_{i-1} = S_1/S_3$ then for an observed $\Delta_i = (-1/0/1)$, for these cases the table entries as in Table I are empty, and we cannot decide upon the subsequent bit. So a solution which already appears in Algorithm 3 can be adopted for $St_{i-1} = S_1/S_3$ since it will make the attack algorithm easier. Since we cannot decide upon the subsequent bit when $St_{i-1} = S_1/S_3$.

- The characteristic property for $St_{i-1} = S_1/S_3$ as in Property 1 is $b_{i-2} = P_{i-1} = P_i \neq b_{i-1}$.

Thus if we inject a fault at $(i-1)^{th}$ position then b_{i-1} gets complemented. Effectively, if $St^K_{i-1} = S_1$ previously then after fault $St^{F_{i-1}}_{i-1}$ becomes S_0. Similarly, if $St^K_{i-1} = S_3$ previously then after fault $St^{F_{i-1}}_{i-1}$ becomes S_2.

- Thus if $St_{i-1} = S_1/S_3$, and we denote the number of branch mispredictions for secret exponent with fault at $(i-1), (i+1)^{th}$ position as bm_0.
- And bm_1 = number of branch mispredictions for secret exponent with fault at $(i-1), i, (i+1)^{th}$ position.
- we define, $\Delta_{i-1,i,i+1} = bm_1 - bm_0$ as the difference in branch misses between a 3-bit and a 2-bit faulty key.
- Following the analysis in the previous subsection we can intuitively write,

a) **If** $St^K_{i-1} = S_1$

In this case the difference of branch miss of the three bit fault will have a similar classification as the two bit in the previous subsection as,

- $\Delta_{i-1,i,i+1} = bm_1 - bm_0$ will behave equivalently to $\Delta_{i,i+1}$(for $St^K_{i-1} = S_0$)
  - If $(\Delta_{i-1,i} = 1)$, then $(b_i, b_{i+1}) = (0, 0)$.
  - If $(\Delta_{i-1,i} = -1)$, then $(b_i, b_{i+1}) = (1, 0)$.
  - If $(\Delta_{i-1,i} \neq 1/-1)$ & $(\Delta_{i-1,i,i+1} = 1)$, then $(b_i, b_{i+1}) = (0, 1)$.
  - If $(\Delta_{i-1,i} \neq 1/-1)$ & $(\Delta_{i-1,i,i+1} = -1)$, then $(b_i, b_{i+1}) = (1, 1)$.

b) **If** $St^K_{i-1} = S_3$ Similarly,

- $\Delta_{i-1,i,i+1} = bm_1 - bm_0$ will behave equivalently to $\Delta_{i,i+1}$(for $St^K_{i-1} = S_2$)
  - If $(\Delta_{i-1,i} = 1)$, then $(b_i, b_{i+1}) = (1, 1)$.
  - If $(\Delta_{i-1,i} = -1)$, then $(b_i, b_{i+1}) = (0, 1)$.
  - If $(\Delta_{i-1,i} \neq 1/-1)$ & $(\Delta_{i-1,i,i+1} = 1)$, then $(b_i, b_{i+1}) = (1, 0)$.
  - If $(\Delta_{i-1,i} \neq 1/-1)$ & $(\Delta_{i-1,i,i+1} = -1)$, then $(b_i, b_{i+1}) = (0, 0)$.

This reduction is performed with two step measurement. Before observing the difference in branch misses, the state of the recovered bits are identified. Then by performing a single

or two bit fault (when in state S_0/S_2) or performing a two and three bit fault (when in state S_1/S_3) the entire identification of the key can be performed. The Algorithm 4 provides an iterative algorithm using the combination of one and two bit faults.

Now if the adversary aims to reveal secret key bits using data from performance counters, he has to correctly classify and identify only the distributions corresponding to 1 and -1. If we observe the individual distributions in Figure 5 more closely, the distributions for $\Delta_i = 1, -1$ are distinct as shown in Figure 6, and the mean values of difference in branch miss obtained from hardware performance counters for the respective distributions are $1.5848, -1.7827$. The maximum and minimum mean values are observed for $\Delta_i = 1$ and -1 respectively. Thus we observe that the distributions corresponding to $\Delta_i = 1$ and -1 are the distributions which are most separated among the other distributions.

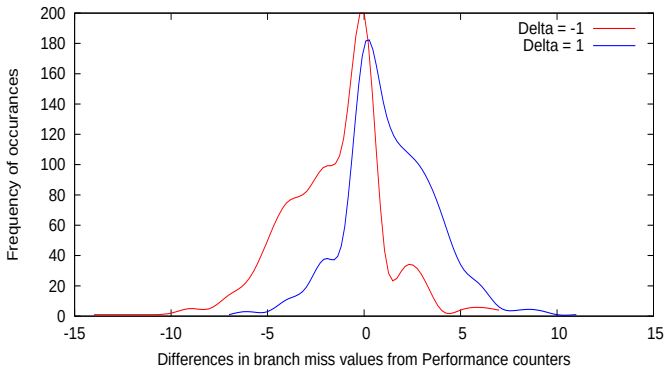

Fig. 6. Distribution of Difference in branch misses between the secret and i^{th} bit faulty key from performance counters on Intel Core i5 for $\Delta_i = 1/-1$

On the basis of this observation, in the next section we provide the detailed results on the classification of differences of branch misses as obtained from hardware performance counters.

VII. Validation of Attack Algorithm using data from Performance Counters

In this section we provide the classification of the key sequences on the basis of the performance counter values. The same experiment as in Section V-1 is performed in this time with the knowledge of the states in the predictor algorithm. This time the distribution of differences are separately constructed for the states S_0, S_1, S_2, S_3. The keys for a state S_j, $0 \leq j \leq 3$ are chosen such that for a pair of secret and its faulty key differing at their i^{th} bit, the state of the 2-bit predictor upto $(i-1)$ bits is in S_j. Table III, IV shows the respective mean values from the distribution of differences in branch misses as obtained from the hardware performance counters on Intel Core i5 and Intel Core 2 Duo platforms.

The entries in Table III for $\Delta_i = 1, -1$ for states S_0, S_2 are highlighted in order to show that the mean values obtained from distribution of differences in branch misses from hardware performance counters have a strong correlation with the theoretical Δ_i values. The means for $\Delta_i = 1, -1$ are having the maximum and minimum value in both the cases. But (for states S_0, S_2) if $\Delta_i = -3, -2, 0, 2, 3$ the distributions overlap to a great extent and we are unable to clearly distinguish the classes from the observed means (as well as their dis-

Algorithm 4: Attack Algorithm Combining One and two bit faults

Input: Unknown key(k) of n bit $(k_0, k_1, \cdots, k_{n-1})$
Output: Retrieved key bits $(k_0, k_1, \cdots, k_{n-1})$
begin
  Assume $k_0 = 1$;
  for $i = 1$ *to* $n-1$ **do**
    St_{i-1} = state of branch predictor for known $i-1$ bits;
    if *(*$St_{i-1} = S_0$ *or* $St_{i-1} = S_2$*)* **then**
      bm_0 = number of branch misses for secret key;
      bm_1 = branch misses for the secret key with fault at i^{th} bit;
      $\Delta_i = bm_1 - bm_0$;
      if *(*$\Delta_i = 1$ *or* $\Delta_i = -1$*)* **then**
        Access Table I at corresponding difference location Δ_i for S_0/S_2;
      else
        bm_0 = branch misses with fault at $(i+1)^{th}$ bit;
        bm_1 = branch misses with fault at i, $(i+1)^{th}$ bit;
        $\Delta_{i,i+1} = bm_1 - bm_0$;
        Access Table II at corresponding difference location $\Delta_{i,i+1}$ for S_0/S_2;
      end
    end
    if *(*$St_{i-1} = S_1$ *or* $St_{i-1} = S_3$*)* **then**
      bm_0 = branch misses for secret key with fault at $(i-1)^{th}$ location;
      bm_1 = branch misses for the secret key with fault at $(i-1), i^{th}$ bit;
      $\Delta_{i-1,i} = bm_1 - bm_0$;
      if *(*$St_{i-1} = S_1$*)* **then**
        Set $St_{i-1} = S_0$;
      else
        Set $St_{i-1} = S_2$;
      end
      if *(*$\Delta_{i-1,i} = -1$ *or* $\Delta_{i-1,i} = 1$*)* **then**
        Access Table I corresponding to S_0 or S_2 at location $\Delta_{i-1,i}$;
      else
        bm_0 = number of branch misses for secret key at $(i-1), (i+1)^{th}$ location;
        bm_1 = branch misses for the secret key with fault at $(i-1), i, (i+1)^{th}$ bit;
        $\Delta_{i-1,i,i+1} = bm_1 - bm_0$;
        Access Table II corresponding to S_0 or S_2 at location $\Delta_{i-1,i,i+1}$;
      end
    end
  end
  Update the partial known key to $(k_0, k_1, \cdots, k_i)$.
end

TABLE III
Tabulating means from hardware performance counters on **Intel Core i5** for specific $St_{i-1} = S_j$ with respective Δ_i's for key length $n = 21$ bit and $i = 10, j = 0, 1, 2, 3$

St_{i-1}	Δ_i						
	-3	-2	-1	**0**	**1**	**2**	**3**
All states	-0.3199	1.3183	-1.7827	0.0004	1.5848	-1.1061	0.3104
S_0	-0.4968	0.1888	-1.6513	0.1138	1.6331	-0.2215	0.2891
S_1	-0.4304	1.8109	-1.7846	-0.1286	1.2348	-1.4589	0.6358
S_2	-0.3091	0.1053	-1.5883	0.1650	1.6752	0.2152	0.2686
S_3	-0.3942	1.8828	-1.7604	-0.0113	1.0932	-1.8714	0.2834

TABLE IV
TABULATING MEANS FROM HARDWARE PERFORMANCE COUNTERS ON **Intel Core 2 Duo** FOR SPECIFIC $St_{i-1} = S_j$ WITH RESPECTIVE Δ_i'S FOR KEY LENGTH $n = 21$ BIT AND $i = 10, j = 0, 1, 2, 3$

St_{i-1}	Δ_i						
	-3	-2	-1	**0**	**1**	**2**	**3**
S_0	0.2201	1.4601	-1.6699	0.0846	1.7751	-1.0907	-0.3539
S_1	-0.085	1.3891	-1.6401	-0.0608	1.3577	-1.4847	0.2337
S_2	0.1584	0.9915	-1.5393	0.3494	1.7987	-0.9951	-0.0741
S_3	0.145	1.9085	-1.5018	0.145	0.9813	-2.4997	0.0741

tributions). Thus we conclude that we can clearly distinguish the distributions with $\Delta_i = 1/-1$ from the remaining in their learning phase. The remaining Δ_i's cannot be uniquely identified only by observing the difference distribution from one-bit fault model. For this reason we perform a similar profiling with the two bit fault model where $\Delta_{i,i+1}$ can assume only two values $1, -1$. This classification is easier since there exists only two classes.

On the contrary when $St_{i-1} = S_1/S_3$, the means corresponding to all possible Δ_i values (for a set of random keys and their i^{th} bit faulty counterparts) in Tables III, IV shows that in this case it is even hard to distinguish cases $\Delta_i = 1, -1$ from the remaining. Thus from this, we conclude that only if $St_{i-1} = S_0/S_2$ and if $\Delta_i = 1, -1$ for a secret key and its i^{th} bit faulty counterpart from 2-bit predictor algorithm, then adversary will succeed to determine the subsequent bits by observing the distribution of differences in branch misses. If $\Delta_i \neq 1, -1$ then the classification can be uniquely performed with $\Delta_{i,i+1}$ values.

In the next subsection we explain the classification of distribution of branch misses for a non-classified secret key using template building and template matching techniques. The mean and variances observed in the learning phase, from the set of known keys for known previous states are used as templates for the template building phase of template attacks.

A. Template Attacks

The template attack [4], [12] is a statistical attack strategy exploiting the inherent properties of observed sample distributions. The attack is performed in two phases: template building and template matching. In the template building procedure, samples (differences of branch misses) are collected using performance counters for a specific key and its faulty counterpart and a distribution is constructed out of the observed samples. Distributions of a set of keys belonging to a specific class is observed and the statistics like mean and variance are learnt in this phase to constitute a template for the corresponding distribution. In the next phase, unknown template such as distribution of correct key is obtained and is compared with each of the learnt templates. The unknown key is supposed to exhibit the maximum probability of matching with its actual correct class. Thus we perform the template matching phase on the template learnt as in Table III. We demonstrate the results for a simple example on 3 different keys where the secret key is of size $n = 21$ bits, the target bit was the 11^{th} bit and the state of the previous 10 bits is taken as S_0. We provide a tabular representation as in Table V of the cumulative probabilities over observed samples for matching a distribution of branch misses for the unknown exponent (from hardware performance counters) belonging to the templates built at the learning phase on **Intel Core i5** with $\Delta_i = 1, -1$ for key length $n = 21$ bit. From the Table V it is clear that

TABLE V
TABULATING SUM OF PROBABILITIES OBSERVED FOR MATCHING A DISTRIBUTION OF BRANCH MISSES FOR AN UNKNOWN EXPONENT

Δ_i of K	Templates for Δ_i						
	-3	-2	-1	**0**	**1**	**2**	**3**
1	38.991	43.638	27.827	40.15	51.053	31.869	46.138
1	53.079	61.159	38.112	52.848	64.368	46.443	58.542
-1	58.228	44.542	65.853	54.432	33.155	62.212	47.392

the Δ_i values as observed from the 2-bit predictor algorithm is showing the maximum probability in each of the cases correctly. Thus if a classification can be done successfully, then it is capable of retrieving the immediate next bits. Thus the iterative algorithm ef

e(A1.49i)0.92566 3

From the table it follows that $\Delta_i = 1, 2, 3$ can classified together, and similarly $\Delta_i = -1, -2, -3$ can be classified together for states S_0/S_2. These classification uniquely retrieves the i^{th} bit as b_i's for these two classes are same. From Table I, if $\Delta_i = 1, 2, 3$ and $St_{i-1} = S_0$, then $b_i = 0$ and for $\Delta_i = -1, -2, -3$ and $St_{i-1} = S_1$ then $b_i = 1$. For the case where $\Delta_i = 0$, the differences from the two bit fault can uniquely retrieve the bits subsequently. Thus like square and multiply algorithm, the Montgomery Ladder implementation can also be foiled using data from performance counters and thus leaking the secret key bits.

B. Attacks on RSA-CRT countermeasures using data from Hardware Performance Counters

The RSA-CRT algorithm as explained in Section II-D is implemented by calculating the CRT exponents as $d_p = d \ mod \ (p-1)$ and $d_q = d \ mod \ (q-1)$ and the Chinese Remainder Theorem (CRT) combines the output of the computation after the exponentiations are performed using this exponents d_p and d_q. The exponentiation operation being performed in RSA-CRT algorithm uses the standard underlying exponentiation algorithms such as square and multiply and Montgomery ladder implementations. A probabilistic $poly(\log N)$ time Algorithm to factorize Modulus N [11] 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. In most of the fault attack countermeasures the output of the faulty computation is fuzzied to some extent so that the difference of the correct and the incorrect computation cannot be utilized by an adversary to factorize the large primes p, q. But in all of these countermeasures, the correctness check is done after the exponentiation on the faulty exponent. So in all of the countermeasures while the exponentiation operation is being performed on the secret as well as the faulty exponent, the side channel leakage through branch miss event can be utilized to reveal secret exponents d_p and d_q with the analysis and the algorithm provided in the paper. Thus the fault attack featuring differences of branch misses by modelling the 2-bit predictor algorithm can be utilized to a great extent to attack the square and multiply, Montgomery ladder and RSA-CRT Algorithms.

VIII. Conclusion

This paper demonstrates a fault attack on exponentiation algorithms featuring the branch misses obtained through the hardware performance counters by modeling the underlying branch predictor unit. The paper demonstrates that the differences of branch misses between a faulty and a correct computation is enough to retrieve the subsequent secret exponent bits. A detailed analysis on the key space reduction by modeling 2-bit dynamic predictor has been provided in the paper. An iterative algorithm retrieving subsequent key bits is developed using the observations. Finally, the attack is demonstrated on various Intel platforms such as Core 2 Duo, Core Intel i3 and Core Intel i5 with the differences of branch misses as observed from the hardware performance counters on actual systems. The classification is performed through a template building and template matching. Templates of differences in branch misses are constructed for some randomly generated keys, and in the attack phase, a secret key distribution is observed to classify correctly with maximum probability leading to the retrieval of subsequent key bits. The attack shows that using the fault attack model featuring branch predictors one can attack implementations of exponentiation: both square and multiply, and Montgomery ladder, and the countermeasures for RSA-CRT implementations which forms the central algorithm for several standard public key ciphers. The work raises the open question on implementation of ciphers on systems with such side channel sources.

References

[1] 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.

[2] Sarani Bhattacharya, Chester Rebeiro, and Debdeep Mukhopadhyay. Hardware prefetchers leak: A revisit of svf for cache-timing attacks. In *MICRO Workshops*, pages 17–23. IEEE Computer Society, 2012.

[3] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[4] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[5] 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.

[6] Agner Fog. The Microarchitecture of Intel and AMD CPU's, An Optimization Guide for Assembly Programmers and Compiler Makers, 2009.

[7] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, 4th Edition*. Morgan Kaufmann, 2006.

[8] Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.

[9] Chong Hee Kim and Jean-Jacques Quisquater. Fault attacks for crt based rsa: New attacks, new results, and new countermeasures. In Damien Sauveron, Constantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, editors, *WISTP*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2007.

[10] 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.

[11] Subhamoy Maitra and Santanu Sarkar. On deterministic polynomial-time equivalence of computing the crt-rsa secret keys and factoring. *IACR Cryptology ePrint Archive*, 2009:62, 2009.

[12] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

[13] 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.

[14] Xueyang Wang and Ramesh Karri. Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In *DAC*, page 79. ACM, 2013.