Zhao LT, Hou R, Wang K *et al.* A novel probabilistic saturating counter design for secure branch predictor. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(5): 1022–1036 Sept. 2021. DOI 10.1007/s11390-021-1253-8

# A Novel Probabilistic Saturating Counter Design for Secure Branch Predictor

Lu-Tan Zhao[1,2], Rui Hou[1,2,*], *Senior Member*, *CCF*, Kai Wang[3], Yu-Lan Su[1,2], Pei-Nan Li[1,2] and Dan Meng[1,2], *Senior Member*, *CCF*

[1] *State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences Beijing 100093, China*

[2] *School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100093, China*

[3] *Department of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China*

E-mail: {zhaolutan, hourui}@iie.ac.cn; wk1220ym@163.com; {suyulan, lipeinan, mengdan}@iie.ac.cn

Received December 31, 2020; accepted August 29, 2021.

**Abstract** In a modern processor, branch prediction is crucial in effectively exploiting the instruction-level parallelism for high-performance execution. However, recently exposed vulnerabilities reveal the urgency to improve the security of branch predictors. The vital cause of the branch predictor vulnerabilities is that the update strategy of the saturating counter is deterministic. As a fundamental building block in a modern branch predictor, previous studies have paid too much attention to the performance and hardware cost and ignored the security of saturating counter. This leaves attackers with the opportunities to perform side-channel attacks on the branch predictor. This paper focuses on the saturating counter to explore a secure and lightweight design to mitigate branch predictor side-channel attacks. Instead of applying the isolation mechanism to branch predictor resources, we propose a novel probabilistic saturating counter design to confuse the attacker's perception of the victim's behaviour. It changes the conventional deterministic state transition function to a probabilistic state transition function. When a branch is committed, the conventional saturating counter needs to be updated about whether the prediction results are correct or not. While for the probabilistic saturating counter, the branch predictor determines whether the update is performed based on the update probability. The probabilistic saturating counter dramatically reduces the ability of the attacker to spy the saturating counter's state. Our analyses using a cycle-accurate simulator suggest that the proposed mechanism incurs 2.4% performance overhead and hardware cost while providing strong protection.

**Keywords** branch predictor, side-channel attack, saturating counter

## 1 Introduction

Branch prediction is the fundamental technique to improve the instruction-level parallelism in modern high-performance processors [1–3]. However, the security vulnerabilities exposed in recent years reveal that there are severe risks in modern branch predictor designs [4–10]. Attackers exploit these vulnerabilities to obtain the branch history (i.e., taken or not-taken branches) and reveal the fine-grained execution traces of the process in terms of basic blocks. For example, BranchScope [4] and Bluethunder [8] attacks reveal sensitive information by detecting a specific branch predictor entry. Furthermore, BranchShadowing [5] also uses the shared branch prediction histories to infer fine-grained control flow in Intel SGX. Because the modern branch predictor resources are shared between different threads, in either a single-core processor or an SMT (simultaneous multithreading) processor, it gives an attacker the opportunity to maliciously perceive the branch history information across different processes and privileges.

Regular Paper

Special Section of APPT 2021

This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant No. XDC02010200 and the National Natural Science Foundation of China under Grant No. 62125208.

*Corresponding Author

©Institute of Computing Technology, Chinese Academy of Sciences 2021

Existing hardware approaches to mitigating branch predictor vulnerabilities focus on isolation, including physical isolation and logical isolation. For physical isolation, BRB [11] is a state-of-the-art implementation that provides individual history tables for different programs. Although BRB tries to limit hardware cost, it is in general impractical to assign separate tables to all thread-privilege level combinations. Methods of logical isolation include encryption and flush. Encryption prevents the information of the previous thread from being perceived by subsequent threads [3, 12, 13], which greatly increases the attacker's difficulty in obtaining the information, and changes the key timely to prevent the side-channels from being constructed. The flushing predictor is known to be effective mechanisms, but its performance overhead is significant with software implementation [9]. Even with the optimized hardware implementation, it still suffers significant performance degradation in an SMT processor [13].

The vital cause of the branch predictor vulnerabilities is that the update strategy of the saturating counter is deterministic. This makes it easy for an attacker to control the saturating counter's state to construct a side-channel attack. As the fundamental building block of branch predictors, the saturating counter provides an excellent cost-efficient way of reducing the penalty due to conditional branches and is widely used in various branch prediction designs from the simple GShare predictor to the latest TAGE-type predictor [14–21]. However, previous studies have paid too much attention to the performance and hardware cost and ignored the security of the saturating counter. The emergence of these security vulnerabilities reminds us to reconsider the design of this critical building block. Therefore, this paper focuses on the saturating counter and explores a secure and lightweight design for mitigating branch predictor vulnerabilities. Overall, this paper makes the following contributions.

1) We investigate the side-channel attacks on the saturating counter and reveal how the saturating counter update strategy affects the information leakage via the saturating counter side channel.

2) We propose a novel probabilistic saturating counter design. It changes the conventional deterministic state transition mode to a probabilistic state transition mode. The probabilistic saturating counter greatly reduces the ability of the attacker to perceive the saturating counter's state.

3) We present a detailed evaluation of our proposed design. The proposed probabilistic saturating counter is modelled and evaluated on a cycle-accurate Gem5-based simulator, and its performance impacts are investigated for different branch predictors, including the state-of-the-art TAGE-SC-L predictor [19].

In the following, we introduce the saturating counter and analyze the existing attacks on branch predictors (Section 2), discuss the threat model (Section 3), introduce the defense strategy (Section 4), implement the countermeasure on modern predictor (Section 5), analyze the security of probabilistic saturating counter (Section 6), evaluate the performance impacts (Section 7), summarize related work (Section 8), and conclude this paper (Section 9).

## 2 Background

The conventional branch predictor design allows different processes to access the same hardware resources for branch prediction directly. The attacker process can influence predictor entries shared with the victim process to spy on the execution of sensitive branches. Additionally, the attacker can achieve malicious training to influence the victim's (speculative) execution [4, 8, 22–24], which, in turn, enables or exacerbates the victim's information leak.

### 2.1 Saturating Counter

Branch predictors are usually composed of various saturating counters which record the branch history. A saturating counter is a finite state machine (FSM) that consists of a set of states, a start state, an input alphabet, and a transition function that maps an input symbol and current state to the next state. A Moore machine extends this with output on each state. Four values define a conventional saturating (CS) counter (saturation threshold, correct increment, wrong decrement, and prediction threshold). Taking the two-bit FSM as an example, it has four states as shown in Fig.1: SN (strongly not taken), WN (weakly not taken), WT (weakly taken) and ST (strongly taken). The most significant bit of the saturating counter predicts the direction of the branch, and the other bits provide hysteresis, thereby the branch predictor requires two successive mispredictions to change the prediction of direction. If one considers the case where the alphabet and output symbols are constrained to be only $\{0, 1\}$, a finite state machine can be used to generate taken/not-taken predictions. Fig.1 and Fig.2 are two typical saturating counter forms [25]. Studies [26] have indicated that better results can be obtained by using two or

more bits to represent the history of each branch, with the cost-effectiveness diminishing rapidly beyond three bits. Therefore, we consider only these two types in our design.

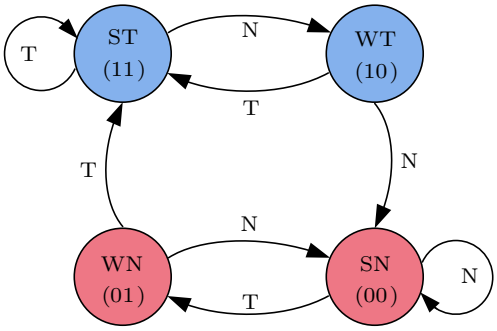



Fig.1. A saturating counter to predict the branch direction.

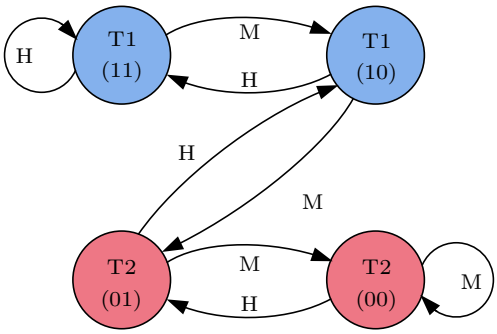



Fig.2. A saturating counter to select a branch predictor table.

There are some other functionally equivalent encoding methods. These saturating counter designs take different forms per the purpose of their use. We classify saturating counters into the following categories according to their functions.

1) The prediction counter predicts the direction of the branch instruction, as shown in Fig.1. The counter is incremented when the branch is taken (T) and decremented with not-taken (N). When the two-bit counter has a value less than or equal to 1, the branch is predicted as not-taken.

2) The choice counter selects a predictor table (T1 or T2), as shown in Fig.2. Some branch predictor architectures have several prediction tables, and a choice table is employed to pick which predictor table to use [27].

3) The confidence counter accesses the quality of prediction. The confidence bit favors the replacement, update, and allocation policy to decide whether to invert a prediction or update a misprediction and allocate a new entry.

4) The weight counter is trained to recognize patterns or correlations among branches. Furthermore, the weight and an input vector are dot producted and thresholded to make a prediction.

We show in Table 1 the types of saturation counters used in basic branch predictors. Note that this table only lists the basic branch predictors. Furthermore, these basic branch predictors also constitute complex branch predictors (e.g., TAGE-SC-L predictor).

**Table 1**. Saturating Counter Types Used in Different Basic Branch Predictors

| Predictor Type | Saturating Counter Type |
|---|---|
| GShare [14] | Prediction counter |
| Bimode [15] | Prediction counter, choice counter |
| Tournament [16] | Prediction counter, choice counter |
| TAGE [21, 28] | Prediction counter, confidence counter |
| Loop predictor [29] | Confidence counter |
| Perceptron [17, 18] | Weight counter |

## 2.2 Prime+Probe Attack on Saturating Counter

Branch predictor side-channel attacks require the critical capability to prime and probe saturating counters. The saturating counter encodes the execution history information of branch instructions, which may also contain sensitive branch instructions. The attack is the process of decoding the information in the saturating counter. Since the state transition of a conventional saturating counter is deterministic, it is easy for an attacker to infer the direction of the victim's branch by counting the branch prediction results (correct prediction or misprediction).

As shown in Fig.3, the attacker process and the victim process share a saturating counter. The attacker first primes the target saturating counters to an initial state (e.g., strong taken) with successive taken branches and then waits for the victim to execute. Victim's branch execution result depends on the secret, which may be taken or not-taken. To distinguish the execution result of the victim's branch, the probe vector used for spy must be the opposite of the prime vector used for initialization (successive not-taken branches). The attacker judges the prediction result of the branch predictor by measuring the execution time of probe process (e.g., reading rdtsc/rdtscp register or related hardware performance counter in x86 ISA). The key to distinguish the victim's behaviors is observable differences of prediction results in the probe process measurements. We note that there was a special point in the detection process. Before this point, the attacker observed

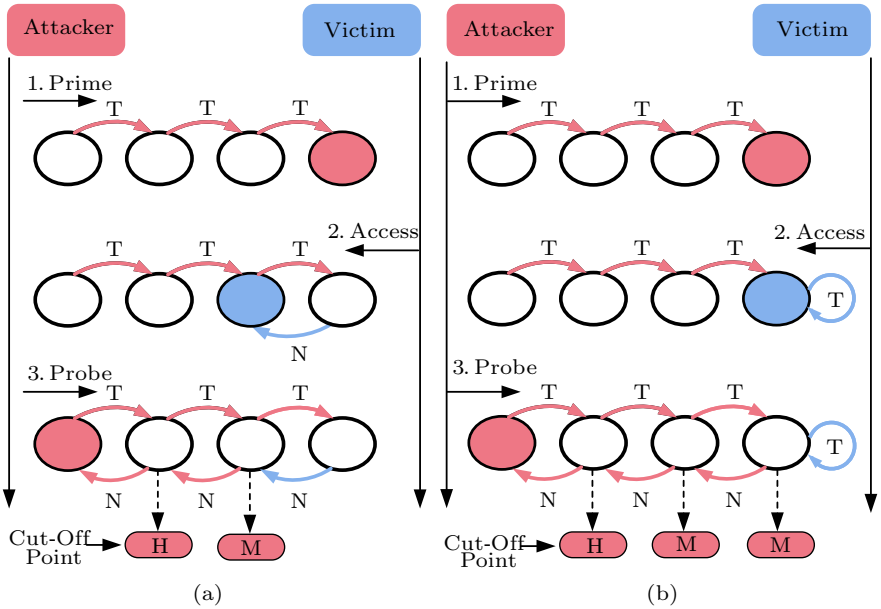



Fig.3. Demonstration of prime+probe attack infers (a) the victim executes upon a not-taken (N) branch and (b) the victim executes upon a taken (T) branch on a two-bit saturating counter. H means correct prediction and M means misprediction.

mispredictions. After this point, all is correct prediction regardless of whether the target branch performed taken or non-taken. The victim's behaviour will be indistinguishable. We call this point as the cut-off point. Therefore, we summarize two conditions for performing a prime+probe attack on the saturating counter.

• The prime vector and the probe vector are in opposite directions.

• The branch prediction results before the cut-off point are distinguishable.

Taking BranchScope [4] for example, an attacker first locates the shared predictor entry of the secret-dependent branch of the victim and sets its saturating counter to a specific state, such as strong taken. The contents of the branch predictor are updated after the victim's target branch instruction is executed. After switching back to the attacker's program, the victim's update to the branch predictor manifests as a measurable difference in the execution time. In BranchScope, the second measurement of the probe process is considered as a cut-off state. The attacker can thus sense the direction of the target instruction and infer the victim's execution path.

## 3 Threat Model and Attacker Capabilities

This paper has the following assumptions: the attacker thread and the victim thread can run on the same processor core. The attacker knows the source code and the address layout of the victim program. The attacker has the ability to run the victim program in single-step mode, such as manipulating the APIC timer exploited in SGX-Step [30].

This paper focuses on mitigating the branch predictor side-channel attacks via saturating counters, which causes malicious perception across different processes and privileges. Side-channel leakage in the event of a misprediction is included in this paper as well. It should be noted that we consider developers of victim processes as trusted. Injecting trojans in software [31] is outside our scope. In addition, mispredictions are perhaps inevitable; thus this paper does not consider the defense against all speculative execution related vulnerabilities.

## 4 Defense Strategy

In a side-channel attack on branch predictors, an attacker infers on the victim's sensitive branch's direction by detecting the saturating counter's state. The reason for this vulnerability is that the state transition rules of the conventional saturating counter are deterministic. Given the initial state, the final state, and the transfer function of the saturating count, an attacker can easily reverse a possible transition (caused by the victim) by controlling the rest of the intermediate process. Therefore, an effective way to defend against the perception attack is to introduce randomization in the state transition function of the saturating counter, confounding the attacker to the point where he/she cannot accurately infer the victim's branch behaviour.

### 4.1 Probabilistic Saturating Counter

To prevent an attacker from perceiving the state of the saturating counter, we propose a probabilistic saturating counter (PSC). Since the update probability (P) is introduced into the transfer function, each state transition becomes a probabilistic event. When a branch is committed, its execution result is fed back to the branch predictor. After receiving an update request, the conventional saturating counter (CSC) needs to be updated about whether the prediction results are correct or not. While for a probabilistic saturating counter, the branch predictor determines whether the update is performed based on the current update probability. Here, the transfer probability between any two states is denoted as $P_{A\to B}$ ($A$ is the current state, and $B$ is the next state), as shown in Fig.4. For example, $P_{\mathrm{SN}\to\mathrm{WN}}$ or $P_{32}$ represents the probability of transition from the current strong not-take state to the next weak not-taken state caused by a taken branch.

The probabilistic update mechanism has been shown in Fig.5. When the update request is incoming, the probability update controller generates an enable signal for update to decide whether to allow an update. In the controller, the probability is calculated by the random generator (pseudo-random number generator) and the probability threshold which is set in advance. If the generated random number is greater than the probability threshold, the update is allowed. Otherwise, the update will be discarded, and the current state remains unchanged, but the global history is still updated. The probability threshold supports multiple values to control the multiple transfer probabilities between different states. Moreover, the update probability can be controlled by dynamically adjusting the probability threshold using a heuristic policy according to the security requirements of the branch predictor. Here we take the two-bit saturating counter as an example. There are other encoding methods, but the update mechanism proposed in this paper can be easily extended to these encoding methods.

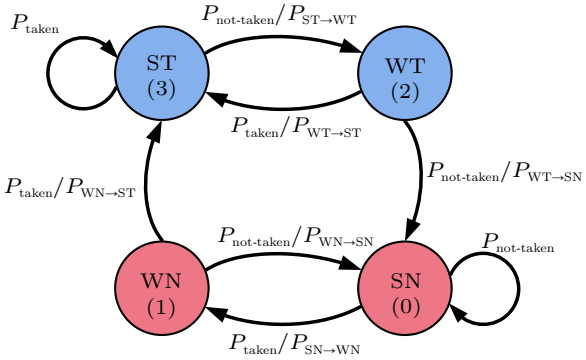



Fig.4. A probabilistic saturating counter with probabilities between any two transition states.

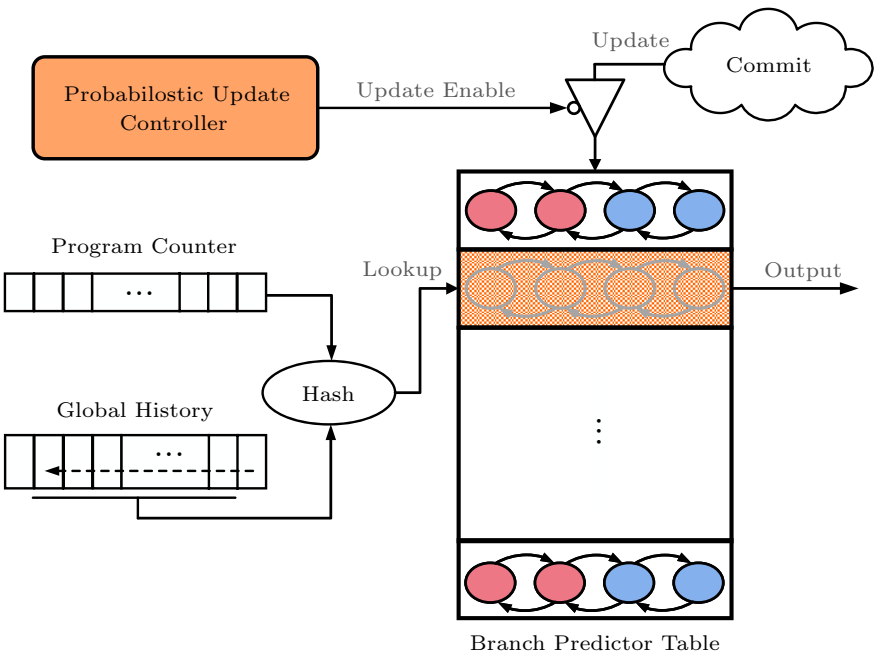



Fig.5. Architecture overview of probabilistic update mechanism.

### 4.2 Transition Function of Probabilistic Saturating Counter

The two-bit saturating counter is a finite state machine, see Fig.1 and Fig.2. FSM can be considered as a Markov chain where the probability of success ($p = P_{\text{taken}}$) is the probability of a branch being taken and the probability of failure ($q = P_{\text{not-taken}} = 1 - P_{\text{taken}}$) is the probability of branch being not taken. The probability of updating saturating counter is $m = P_{\text{current}\to\text{next}}$ and the probability of not updating the saturating counter is $n = 1 - m$ shown in Fig.4. Then, the Markov matrices ($\boldsymbol{M}$) for the conventional saturating counter and the probabilistic saturating counter can be expressed on the left and right sides of the arrow in (1) respectively [32].

$$\begin{pmatrix} p & q & 0 & 0 \\ p & 0 & 0 & q \\ p & 0 & 0 & q \\ 0 & 0 & p & q \end{pmatrix} \Rightarrow \begin{pmatrix} p+nq & mq & 0 & 0 \\ mp & n & 0 & mq \\ mp & 0 & n & mq \\ 0 & 0 & mp & q+np \end{pmatrix}. \quad (1)$$

Given the symmetry of the state transition, the Markov matrix of the probabilistic update saturating counter is a symmetric matrix, where $P_{\text{ST}\to\text{WT}}$ is equal to $P_{\text{SN}\to\text{WN}}$ and $P_{\text{WT}\to\text{WN}}$ is equal to $P_{\text{WN}\to\text{WT}}$. Then $M$ is substituted into (2) to solve the probability of the steady state.

$$\boldsymbol{u} \times \boldsymbol{M} = \boldsymbol{u}, \quad (2)$$

where $\boldsymbol{u} = (x, y, z, w)$ and $x + y + z + w = 1$ . We solve (2) and find that the two Markov matrices have the same characteristic vectors, that is, they have the same steady state. Therefore, the probabilistic update mechanism does not degrade the prediction accuracy of the saturating counter without alias conflict. Note that even so, the training times for the saturating counter becomes uncertain, and when multiple branches share a saturating counter, the number of mispredictions may be increased when alias conflict occurs. And we evaluate the performance impact in detail in Section 7.

## 5 Implementation on Modern Predictors

In this section, we take the latest TAGE-SC-L [19] branch predictor, as an example of how to apply probabilistic saturating counter to modern branch predictors. The TAGE-SC-L predictor features a TAGE predictor, a loop predictor (LP) and a multi-GEHL statistical corrector (SC). The TAGE predictor provides the main prediction. Then this prediction is used by SC whose role consists in confirming or reverting the prediction. The loop predictor is used to predict regular loops with loop bodies.

*Replacing the Prediction Counter with the Probabilistic Saturating Counter.* The TAGE-SC-L branch predictor contains several types of saturating counters. First, each entry of TAGE predictor tables consists of prediction counters which record the direction of branches. Then, due to the limited size of the predictor, the useful bit which is a confidence counter is used to dynamically determine the number of entries to allocate on a misprediction and is reset according to the reset policy. However, no matter what kind of table provides the prediction, it would eventually end up leaking information through a prediction counter. Thus considering the balance of performance and security, it is only necessary to replace the direction prediction counter with the probabilistic saturating counter, without changing the confidence counter. In addition, we do not need to change the original structure of the SRAM macros. Importantly, we deploy a probability generator module into the update path. There is no need to change the original update logic. The original enable signal for update performs the AND operation with the output of the probability generator module which is generated in parallel to the update signal from the commit stage. Thus, the impact of this modification on the critical path is minimal.

*Compatible with Hysteresis Bit Compression.* To make the branch predictor structure more compact, the base predictor employs the hysteresis bit compression technology. Since the probabilistic saturating counter only needs to modify the branch predictor's update logic slightly, there is no loss of the entropy of hysteresis bit. Therefore, these techniques can still be applied to the probabilistic update predictor.

## 6 Security Analysis

The key to the saturating counter side-channel attacks is that the attacker can find a steady cut-off point to distinguish the saturating counter's state clearly. Due to the probabilistic update, the state transition of the saturating counter becomes uncertain. It increases the randomness in the training of the saturating counter. In this section, we analyze how the attacker finds the cut-off point and then evaluate the security of the probabilistic saturating counter.

To simplify the analysis, we assume that the attacker is powerful enough to manipulate the specific branch predictor entry shared with the target branch

in a noise-free environment. When an attacker can find a clearly distinguishable cut-off point, we consider the attack successful. (This is a pessimistic assumption, and in fact, it usually takes a long time in the attack and is a significant factor limiting the bandwidth of the side channel.) To assess the security of probabilistic saturating counter, we design Algorithm 1 to abstract the prime+probe attack on a saturating counter shown in Fig.3. If a branch is taken, but the saturating counter predicts it as not-taken, the attacker will observe a misprediction. Otherwise, the attacker observes a correct prediction. In Algorithm 1, an attacker first trains the saturating counter to a starting state using the prime vector (lines 1–3). The victim process then executes the target branch in a random direction (line 4). In the attack process, the more the times the victim executes the target branch, the more difficult it is for the attacker to distinguish the victim's branch behavior. Therefore, the best choice for the attacker is to ensure that the victim executes the target branch instruction only once at a time. Since the prime vector and the probe vector are in the opposite direction, the first few measurements are mispredictions (as shown by M in Fig.3), and the subsequent ones will always be correct predictions (as shown by H in Fig.3). Finally, the attacker executes the probe vector until a correct prediction is observed (lines 5–11). By analyzing the differences in the number of the observed mispredictions, an attacker can find a clear cut-off point to infer whether the victim's branch is taken or not-taken.

Fig.6(a) shows the relationship between the success rate of the prime+probe attack on the saturating counter and the update probability. We see that the lower the update probability, the stronger the security. When the update probability is 0, there is no branch history stored in the saturating counter and no secret leakage. At this point, the attacker guesses completely randomly, and the probability of success is 50%. The conventional saturating counter has an update probability of 100%; thus it is vulnerable to malicious perception attacks. Furthermore, compared with the three-bit saturating counter and the two-bit saturating counter, it can be found that the three-bit saturating counter has better security strength than the two-bit saturating counter at the same update probability.

**Algorithm 1.** Finding the Cut-Off Point

**Input:** ***Pm***: prime vector (0, 0, 0, ...), $V$: victim (sensitive values), ***Pb***: probe vector (1, 1, 1, ...)
**Output:** the number of probes before the cut-off point

```
1:  for m in Pm do
2:      Execute target branch with direction m
3:  end for
4:  Execute victim branch with direction V
5:  while b in Pb do
6:      Execute target branch with direction b
7:      if prediction is hit then
8:          Return C
9:      end if
10:     C ← C + 1
11: end while
12: Return C
```

Fig.6(b) shows the probability distribution of the number of probes required for the saturating counter changing from the initial state to the cut-off state with an update probability of 50%. The name of the saturating counter is followed by its bit wide and the victim's branch direction in Fig.6(b). For example, PSC2_1 represents that the attack process performs the prime+probe attack on a two-bit probabilistic saturating counter and the victim process executes a taken branch. For the conventional saturating counter, the number of probes is completely concentrated on a certain value without any overlap. Therefore, the attacker can easily find the cut-off point difference to infer the

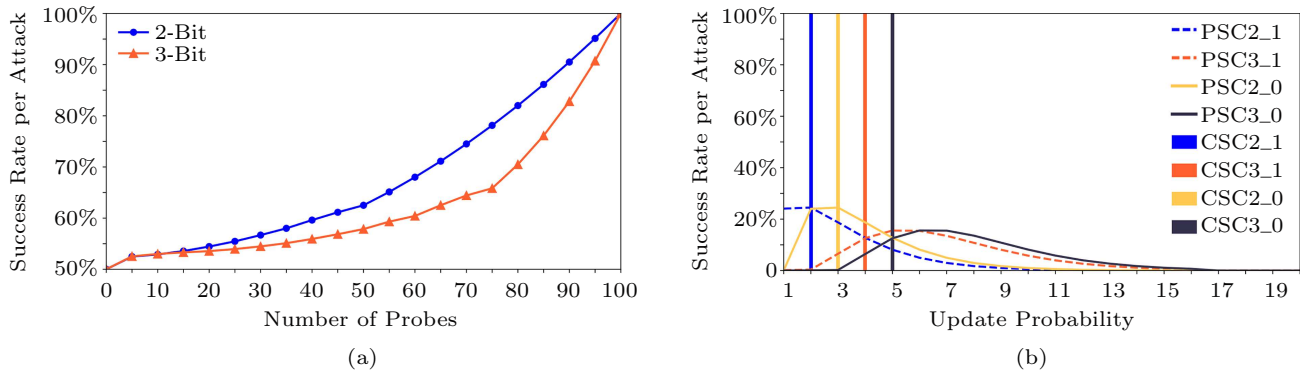



Fig.6. Security analysis of probabilistic saturating counter. (a) The success rate varies with the update probability. (b) Probability distribution of the number of probes.

victim's control flow. However for the probabilistic saturating counter, the number of probes is scattered and the overlapping part of different victims' branch direction increases. And the more the overlapping parts, the harder it is for an attacker to distinguish cut-off points. For a two-bit saturating counter, an attacker can accurately perceive the victim's branch behaviour with a probability of about 63%, and for a three-bit saturating counter, the probability is only 58%, which is close to the probability of no update. These probabilities are much smaller than 100% of the conventional saturating counter. In this case, the probability that an attacker steals 32-bit data continuously is only one in a billion.

The probabilistic saturating counter mitigates the branch predictor side-channel attacks, especially for the one-time attack. However, the probabilistic update policy is based on a probability model. Since the jump direction of a branch instruction has a certain tendency, it is still possible for the attacker to increase the success rate of the attack by repetition. However, repeated attacks face two key challenges that make it difficult to implement. 1) Since the attacker cannot precisely control the execution behavior of the victim process, aligning some of the inferred key bits to restore the complete key information is very difficult. 2) Repeated attacks have obvious behavioral characteristics that can be easily detected by side-channel detection mechanisms [33–35]. These detection methods and our technique are orthogonal; thereby they can be combined to further defend repeated attacks.

## 7 Evaluation

### 7.1 Methodology

We modeled an out-of-order processor using the cycle-level Gem5 simulator [36]. This core is modeled after the latest Intel Sunny Cove core①, and its parameters are shown in Table 2. We experimented on the probabilistic update mechanism with three typical predictors (Gshare [14], Tournament [16], TAGE-SC-L [19]). Probabilistic update saturating counters are implemented in these predictors with an update probability of 50%. We adopted SPEC CPU 2017 benchmark suite [37] with a reference input size for performance evaluation. The simulator was warmed up for one billion instructions, and then ran another billion instructions in the cycle-accurate mode. We measured the misprediction per kilometer instructions (MPKI) to demonstrate the accurate changes of the branch prediction. Moreover, the IPC throughput was measured as the metric of performance. Finally, the probabilistic update mechanism was implemented using Register-Transfer Level (RTL) code and then synthesized with TSMC 28 nm technology for area and timing evaluations.

**Table 2**. OoO Processor Core Configurations

| Parameter | Configuration |
|---|---|
| ISA | ARM |
| Frequency | 2.5 GHz |
| Processor type | 8-decode,8-issue, 8-commit |
| Pipeline depth | 19 stages, fetch 4 cycles |
| ROB/LDQ/STQ/IQ | 352/128/72/120 entries |
| BTB | 1 024 × 4-way entries |
| PHT | TAGE-SC-L: 66.6 KB |
| | or Tournament: 6.3 KB |
| | or GShare: 2 KB |
| ITLB/DTLB | 64/64 entries |
| L1 ICache | 32 KB, 4-way, 64 B line |
| L1 DCache | 48 KB, 4-way, 64 B line |
| L2 Cache | 512 KB, 16-way, 64 B line |
| L3 Cache | 4 MB, 32-way, 64 B line |

### 7.2 Impact on Prediction Accuracy of Different Branch Predictors

Although the probabilistic update enhances the security, the training process of the saturating counter will also become uncertain, which may introduce extra mispredictions. Fig.7 shows the impacts on the prediction accuracy of probabilistic update mechanisms with three different branch predictors (GShare, Tournament and TAGE-SC-L, with a measured baseline MPKI of 11.7, 6.5, and 4.4, respectively). Each bar represents the MPKI of the branch predictors with two- or three-bit saturating counters. The solid part of the bar represents the measured baseline MPKI without any protection (baseline), the lighter part represents the MPKI introduced by the probabilistic update mechanism (probability), and the negative part represents the MPKI is reduced.

Overall, the increase in MPKI is relatively small. The average MPKI growth of the branch predictor with the 2-bit saturating counter is 0.76, 0.45, and 0.54 respectively. And as we expected, the average MPKI growth of the three different branch predictors with

①WikiChip. Sunny cove-microarchitectures-intel, 2019. https://en.wikichip.org/wiki/intel/microarchitectures/sunny_cove, Sept. 2021.

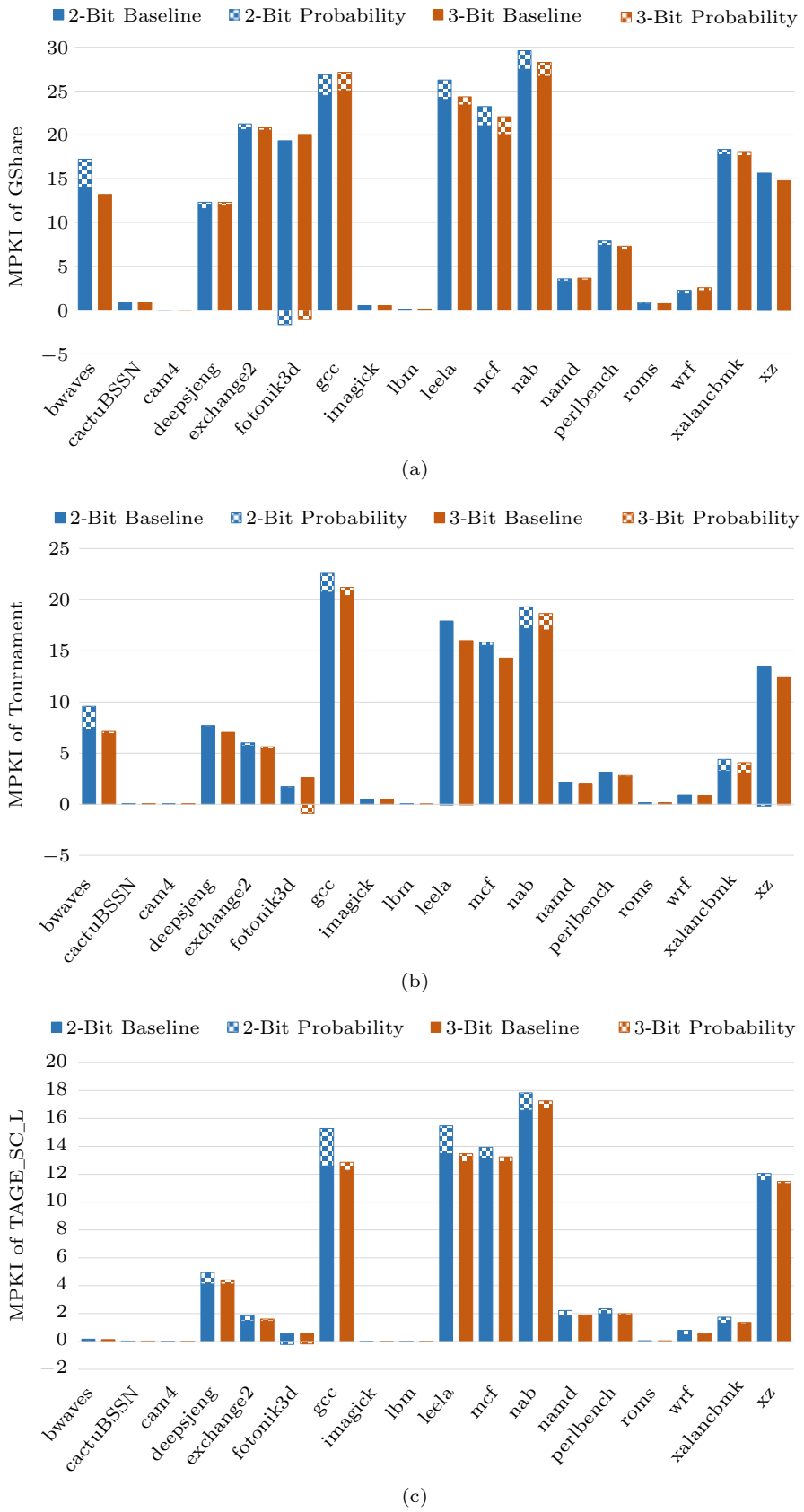



Fig.7. Misprediction per kilometer instruction (MPKI) of different branch predictors (the solid bar represents the MPKI of baseline, and the lighter bar is the additional MPKI introduced by the probabilistic update mechanism). (a) GShare branch preditor. (b) Tournament branch predictor. (c) TAGE-SC-L branch predictor.

the 3-bit saturating counter is less than 0.4. Thus, the branch predictor with 3-bit saturating counter is more tolerant of probability updates than the two-bit branch predictor. In all bars, the most obvious increment occurs in the cases bwaves and gcc, which have increased by 3.2 and 2.4, respectively. Interestingly, there are 8.5% improvements in the prediction accuracy of fotonik3d for GShare, and the improvement of the TAGE-SC-L branch predictor even reaches 30%. Different branch predictors have different sensitivity to different benchmark cases. More complex branch predictors, like TAGE-SC-L, are less affected by the probabilistic update because the newly allocated entries of the TAGE predictor will be initialized as weak taken, thereby reducing mispredictions caused by branch conflict.

### 7.3 Performance Evaluation on Different Branch Predictors

Fig.8 shows the performance impacts of probabilistic update policy mechanisms on three different branch predictors. The performance overhead is compared with the same predictor without any protection. For the example fotonik3d, its performance also has a significant improvement, which indicates that the probability update saturating counter may help improve the accuracy of branch predictors. Thus, we found that the probabilistic saturating counter improves the security of the branch predictor with only a marginal performance loss. Building lightweight secure branch predictors from an update policy perspective is also an effective defense technique. Three observations can be made.

1) There is a trivial range of performance impacts. In some cases, the application's sensitivity to branch predictor behavior and the probabilistic update can result in more than 14% performance degradation. But on average, the performance cost for protection, at a few percent, is quite reasonable.

2) Systems with a more accurate predictor tend to show more performance impact due to protection. However, on average, the increase is not significant: it goes from 2.5% for the least accurate predictor (GShare) to 4.9% for the most accurate one (TAGE-SC-L).

3) Although there are exceptions, in general, predictors with the 3-bit saturating counter incur a lower performance impact than the ones with the 2-bit saturating counter. Compared with the 2-bit counterpart, performance overhead due to the probabilistic update policy is 59%–72% lower.

### 7.4 Sensitivity Analysis of Update Probability

Update probability will affect the training times for the saturating counters. Therefore we take the tournament predictor as an example to evaluate the impact of different update probabilities on the performance because the tournament predictor is almost entirely composed of saturating counters and moderate prediction accuracy.

Fig.9 shows that as the update probability increases, the performance overhead decreases. When the update probability is 10%, the performance overhead is as high as 12.3%. When the update probability increases to 90%, the performance overhead is reduced to 1.9%. Therefore, as we expected, the lower the update probability, the higher the security, but the greater the performance loss.

### 7.5 Power Cost

We modeled energy consumption of an out-of-order processor with the probabilistic update mechanism on the tournament branch predictor using the energy models from prior work [38]. We evaluated the overall processor energy and observed the same trend as the performance overhead result in Fig.9 because of the gradual reduction in execution time. When the update probability changes from 10% to 90%, the energy consumption change fluctuates between +1.9% and −0.6%. In general, the impact of the probability update mechanism on energy consumption is negligible.

### 7.6 Hardware Cost

The TAGE predictor with probabilistic saturating counter is implemented at the register-transfer level (RTL). Based on the TSMC 28 nm technology, we used Synopsys ASIC design flow and synthesis tools to assess the timing and area cost. In the evaluation, to meet most of the design requirements as many as possible, the width of both the random generator and the probability threshold is 16 bits, and the probability accuracy that can be provided is about $2E-05$. It can be seen that the probabilistic update mechanism has minor area and timing cost. For example, in the case of the TAGE predictor with 2-bit saturating counters, the timing cost of probability generation logic is increased by only 2.0% and the area cost is increased by 0.022% (synthesized with TT corner using design compiler).

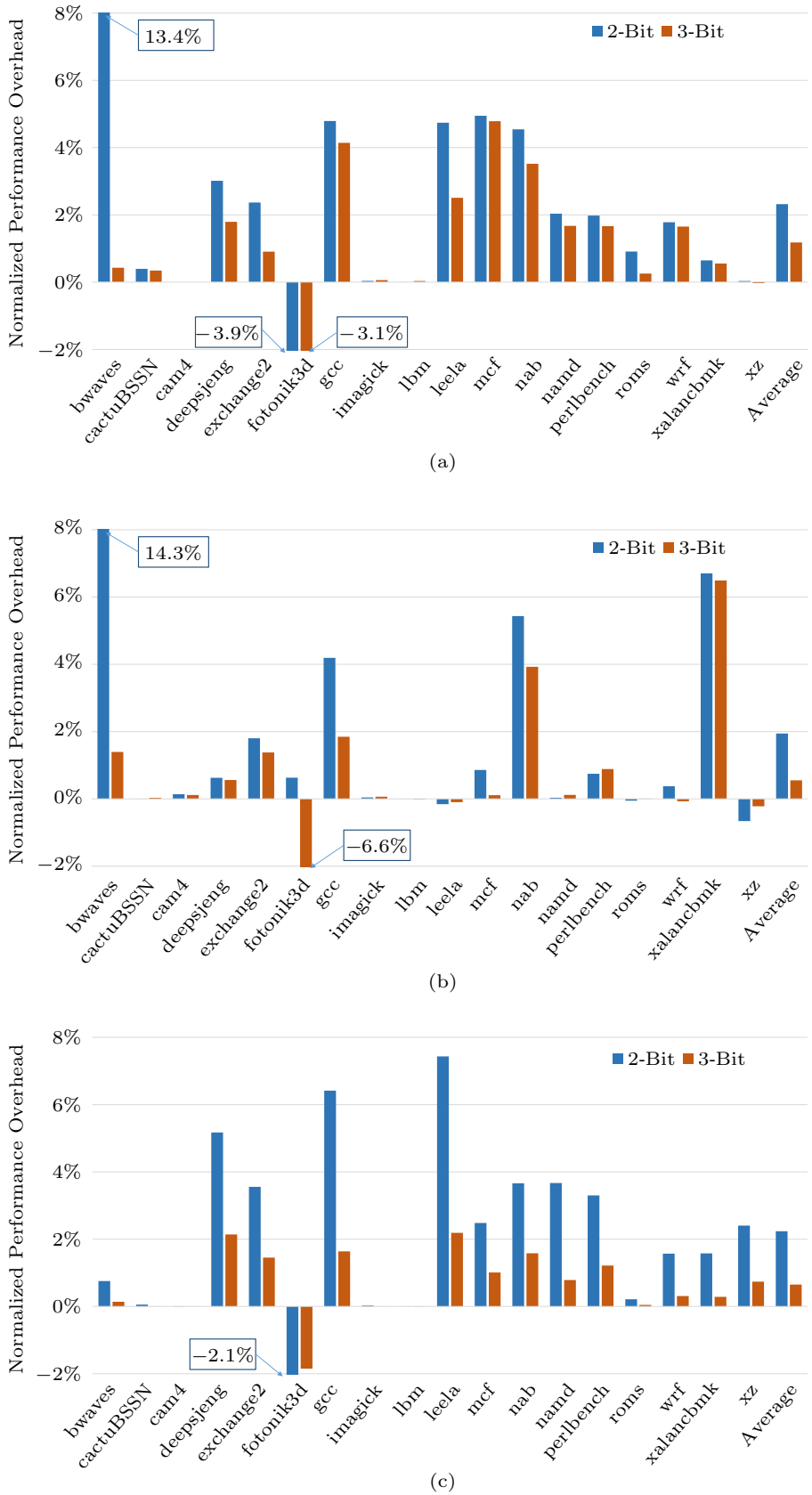



Fig.8. Normalized performance overhead of different branch predictors. (a) GShare branch predictor. (b) Tournament branch predictor. (c) TAGE-SC-L branch predictor.

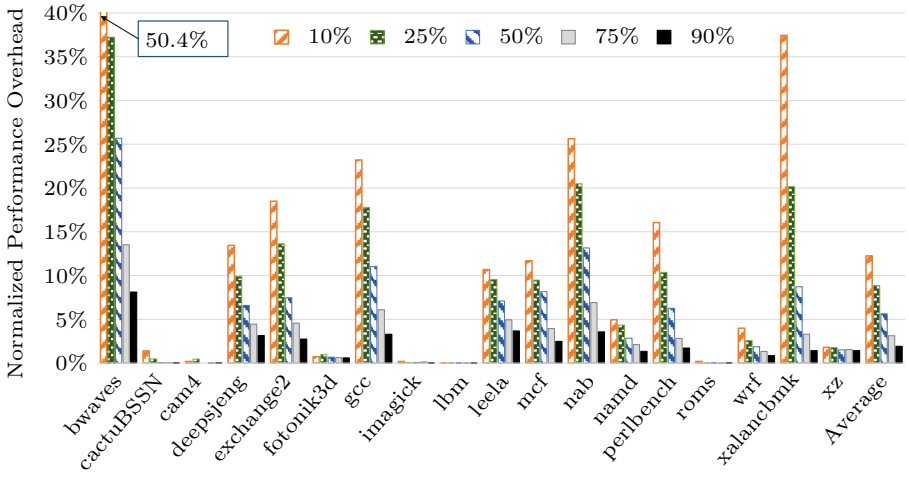



Fig.9. Normalized performance overhead of different update probabilities on a single-threaded core with a tournament branch predictor.

## 8 Related Work

Several countermeasures have been proposed to mitigate the branch predictor side channels. They can be classified into four as follows.

First, branches that include sensitive information can be transformed into safe instructions that do not leave a mark in the branch predictors [39]. Limiting performance counter usage can reduce the information obtained by the attacker②. InvisiSpec [40], Conditional Speculation [41], and STT [42] prevent speculative computation from generating visible microarchitectural state.

Second, the predictor table can be flushed to contain new randomized results. Performing this by software during context switch can bring non-trivial overhead [9]. Such expensive operations can be limited to only the sensitive processes [43]. The impact on the performance and prediction accuracy of flushing predictor tables in hardware has been studied [44]. Not surprisingly, the longer the context switch interval, the smaller the impacts.

Third, using more dedicated hardware is a general approach to isolating states from different processes. For example, sensitive applications in SGX can be allocated with their branch predictor tables [4]. Earlier work on performance improvement considered saving and restoring compressed branch prediction information [45] or providing thread-private branch predictors on SMT processors [46]. BRB is a proposal to retain partial predictor state in on-chip SRAM to swap in with context [11].

Finally, one promising solution is randomizing index and content of branch predictors. The Samsung Exynos CPU has implemented content-encryption via simple substitution cipher in branch-target buffers (BTB) and return-address stack (RAS) [3], but it only protects against some Spectre variants (e.g., Spectre V2 and Spectre RSB), lack of sufficient coverage to side-channel attacks. Lee *et al.* [12] and Zhao *et al.* [13] proposed to randomize the index of branch predictor to mitigate branch predictor side-channels using a low-latency cipher. However, they used the LLBC proposed by CEASER [47], which has been proved to be linear and vulnerable to cryptanalytic attacks, and the complexity of finding an eviction set is the same as when there is no randomization present [48, 49].

## 9 Conclusions

The saturating counter is a fundamental building block in modern branch predictors. However, the security of the saturating counter has been ignored. This leaves the attackers with the opportunities to perform prime-probe attacks on branch predictors. Instead of applying isolation to branch predictor resources, we proposed a novel saturating counter design to confuse the attacker's perception of the victim's behaviour. It breaks the traditional deterministic state transition mode and introduces the probabilistic update mechanism. The probabilistic saturating counter greatly reduces the ability of the attacker to spy the saturating counter state. Our evaluations using a cycle-accurate simulator demonstrated that the proposed mechanisms occur less than 2.4% slowdown on average. Compared

②Guide P. Intel® 64 and ia-32 architectures software developer's manual. Volume 3B: System programming Guide, Part, 2011, 2.

with the isolation mechanism, the probabilistic update mechanism achieves a good balance between performance and safety.

**Acknowledgement(s)** We would like to thank Prof. Naijun Zhan and Prof. Lijun Zhang for their discussions about security analysis of probabilistic saturating counters. We also wish to thank the anonymous reviewers and editors for their valuable comments and suggestions.

## References

[1] Sinharoy B, Van Norstrand J A, Eickemeyer R J *et al.* IBM POWER8 processor core microarchitecture. *IBM Journal of Research and Development*, 2015, 59(1): Article No. 2. DOI: 10.1147/JRD.2014.2376112.

[2] Suggs D, Subramony M, Bouvier D. The AMD "Zen 2" processor. *IEEE Micro*, 2020, 40(2): 45-52. DOI: 10.1109/MM.2020.2974217.

[3] Grayson B, Rupley J, Zuraski G Z *et al.* Evolution of the Samsung Exynos CPU microarchitecture. In *Proc. the 47th ACM/IEEE Annual International Symposium on Computer Architecture*, May 30-June 3, 2020, pp.40-51. DOI: 10.1109/ISCA45697.2020.00015.

[4] Evtyushkin D, Riley R, Abu-Ghazaleh N C, Ponomarev D. BranchScope: A new side-channel attack on directional branch predictor. In *Proc. the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2018, pp.693-707. DOI: 10.1145/3173162.3173204.

[5] Lee S, Shih M W, Gera P *et al.* Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *Proc. the 26th USENIX Security Symposium*, August 2017, pp.557-574.

[6] Aciiçmez O, Koç Ç K, Seifert J P. Predicting secret keys via branch prediction. In *Proc. the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology*, February 2007, pp.225-242. DOI: 10.1007/11967668_15.

[7] Aciiçmez O, Koç Ç K, Seifert J P. On the power of simple branch prediction analysis. In *Proc. the 2nd ACM Symposium on Information, Computer and Communications Security*, March 2007, pp.312-320. DOI: 10.1145/1229285.1266999.

[8] Huo T, Meng X, Wang W *et al.* Bluethunder: A 2-level directional predictor based side-channel attack against SGX. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019, 2020(1): 321-347. DOI: 10.46586/tches.v2020.i1.321-347.

[9] Evtyushkin D, Ponomarev D, Abu-Ghazaleh N. Understanding and mitigating covert channels through branch predictors. *ACM Transactions on Architecture and Code Optimization*, 2016, 13(1): Article No. 10. DOI: 10.1145/2870636.

[10] Bhattacharya S, Mukhopadhyay D. Fault attack revealing secret keys of exponentiation algorithms from branch prediction misses. *IACR Cryptol. ePrint Arch*, 2014, 2014: Article No. 790.

[11] Vougioukas I, Nikoleris N, Sandberg A *et al.* BRB: Mitigating branch predictor side-channels. In *Proc. the 2019 IEEE International Symposium on High Performance Computer Architecture*, February 2019, pp.466-477, DOI: 10.1109/HPCA.2019.00058.

[12] Lee L, Ishii Y, Abu-Sunwoo D. Securing branch predictors with two-level encryption. *ACM Transactions on Architecture and Code Optimization*, 2020, 17(3): Article No. 21. DOI: 10.1145/3404189.

[13] Zhao L, Li P, Hou R *et al.* A lightweight isolation mechanism for secure branch predictors. arXiv:2005.08183, 2020. https://arxiv.org/abs/2005.08183, May 2021.

[14] McFarling S. Combining branch predictors. Technical Report, Digital Western Research Laboratory, 1993. https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf, May 2021.

[15] Lee C, Chen I K, Mudge T N. The bi-mode branch predictor. In *Proc. the 30th Annual International Symposium on Microarchitecture*, Dec. 1997, pp.4-13. DOI: 10.1109/MICRO.1997.645792.

[16] Kessler R E. The Alpha 21264 microprocessor. *IEEE Micro*, 1999, 19(2): 24-36. DOI: 10.1109/40.755465.

[17] Jimenez D A, Lin C. Dynamic branch prediction with perceptrons. In *Proc. the 7th HPCA International Symposium on High-Performance Computer Architecture*, Jan. 2001, pp.197-206. DOI: 10.1109/HPCA.2001.903263.

[18] Tarjan D, Skadron K. Merging path and gshare indexing in perceptron branch prediction. *ACM Transactions on Architecture and Code Optimization*, 2005, 2(3): 280-300. DOI: 10.1145/1089008.1089011.

[19] Seznec A. TAGE-SC-L branch predictors again. In *Proc. the 5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5)*, June 2016.

[20] Seznec A. A 256 Kbits L-TAGE branch predictor. In *Proc. the 2nd JILP Workshop on Computer Architecture Competitions (JWAC-2): Championship Branch Prediction (CBP-2)*, Dec. 2006.

[21] Seznec A. A new case for the TAGE branch predictor. In *Proc. the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2011, pp.117-127. DOI: 10.1145/2155620.2155635.

[22] Kocher P, Horn J, Fogh A *et al.* Spectre attacks: Exploiting speculative execution. In *Proc. the 2019 IEEE Symposium on Security and Privacy*, May 2019, pp.1-19, DOI: 10.1109/SP.2019.00002.

[23] Chen G, Chen S, Xiao Y *et al.* SgxPectre: Stealing intel secrets from SGX enclaves via speculative execution. In *Proc. the 2019 IEEE European Symposium on Security and Privacy*, June 2019, pp.142-157. DOI: 10.1109/EuroSP.2019.00020.

[24] Evtyushkin D, Ponomarev D, Abu-Ghazaleh N. Jump over ASLR: Attacking branch predictors to bypass ASLR. In *Proc. the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, October 2016, Article No. 40.

[25] Ahn Y J, Hwang D Y, Lee Y S *et al.* Saturating counter design for meta predictor in hybrid branch prediction. In *Proc. the 8th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing*, December 2009, pp.217-221.

[26] Lee J K F, Smith A J. Branch prediction strategies and branch target buffer design. *Computer*, 1984, 17(1): 6-22. DOI: 10.1109/MC.1984.1658927.

[27] Sherwood T, Calder B. Automated design of finite state machine predictors for customized processors. In *Proc. the 28th Annual International Symposium on Computer Architecture*, June 30-July 4, 2001, pp.86-97. DOI: 10.1145/379240.379254.

[28] Seznec A, Michaud P. A case for (partially) TAgged GEometric history length predictors. *Journal of Instruction Level Parallelism*, 2006, 8: Article No. 1.

[29] Sherwood T, Calder B. Loop termination prediction. In *Proc. the 3rd International Symposium on High Performance Computing*, October 2000, pp.73-87. DOI: 10.1007/3-540-39999-2_8.

[30] Bulck J V, Piessens F, Strackx R. SGX-Step: A practical attack framework for precise enclave execution control. In *Proc. the 2nd Workshop on System Software for Trusted Execution*, October 2017, Article No. 4. DOI: 10.1145/3152701.3152706.

[31] Zhang T, Koltermann K, Evtyushkin D. Exploring branch predictors for constructing transient execution trojans. In *Proc. the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2020, pp.667-682. DOI: 10.1145/3373376.3378526.

[32] Elkhouly R, El-Mahdy A, Elmasry A. 2-Bit branch predictor modeling using Markov model. *Procedia Computer Science*, 2015, pp.650-653. DOI: 10.1016/j.procs.2016.05.115.

[33] Zhang Y, Juels A, Oprea A, Reiter M K. HomeAlone: Co-residency detection in the cloud via side-channel analysis. In *Proc. the 2011 IEEE Symposium on Security and Privacy*, May 2011, pp.313-328. DOI: 10.1109/SP.2011.31.

[34] Crane S, Homescu A, Brunthaler S *et al.* Thwarting cache side-channel attacks through dynamic software diversity. In *Proc. the 22nd Annual Network and Distributed System Security Symposium*, February 2015. DOI: 10.14722/ndss.2015.23264.

[35] Sabbagh M, Fei Y, Wahl T *et al.* SCADET: A side-channel attack detection tool for tracking Prime+Probe. In *Proc. the 2018 IEEE/ACM International Conference on Computer-Aided Design*, Nov. 2018, pp.1-8. DOI: 10.1145/3240765.3240844.

[36] Binkert N, Beckmann B, Black G *et al.* The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011, 39(2): 1-7. DOI: 10.1145/2024716.2024718.

[37] Bucek J, Lange K D, Von Kistowski J. SPEC CPU2017: Next-generation compute benchmark. In *Proc. the Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, April 2018, pp.41-42. DOI: 10.1145/3185768.3185771.

[38] Li S, Ahn J H, Strong R D *et al.* McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, December 2009, pp.469-480. DOI: 10.1145/1669112.1669172.

[39] Agosta G, Breveglieri L, Pelosi G *et al.* Countermeasures against branch target buffer attacks. In *Proc. the Workshop on Fault Diagnosis and Tolerance in Cryptography*, Sept. 2007, pp.75-79. DOI: 10.1109/FDTC.2007.10.

[40] Yan M, Choi J, Skarlatos D *et al.* InvisiSpec: Making speculative execution invisible in the cache hierarchy. In *Proc. the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, Oct. 2018, pp.428-441. DOI: 10.1109/MICRO.2018.00042.

[41] Li P, Zhao L, Hou R *et al.* Conditional speculation: An effective approach to safeguard out-of-order execution against spectre attacks. In *Proc. the 2019 IEEE International Symposium on High Performance Computer Architecture*, Feb. 2019, pp.264-276. DOI: 10.1109/HPCA.2019.00043.

[42] Yu J, Yan M, Khyzha A *et al.* Speculative taint tracking (STT): A comprehensive protection for speculatively accessed data. In *Proc. the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, October 2019, pp.954-968. DOI: 10.1145/3352460.3358274.

[43] Hu W M. Lattice scheduling and covert channels. In *Proc. the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1992, pp.52-61. DOI: 10.1109/RISP.1992.213271.

[44] Pasricha S, Veidenbaum A. Improving branch prediction accuracy in embedded processors in the presence of context switches. In *Proc. the 21st International Conference on Computer Design*, Oct. 2003, pp.526-531. DOI: 10.1109/ICCD.2003.1240950.

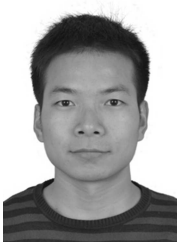
[45] Dhodapkar A S, Smith J E. Saving and restoring implementation contexts with co-designed virtual machines. In *Proc. Workshop on Complexity-Effective Design*, June 2001.

[46] Ramsay M, Feucht C, Lipasti M H. Exploring efficient SMT branch predictor design. http://citeseerx.ist.psu.edu/viewdoc/citations;jsessionid=156C5BEB0B1C452690D-8A3BBE301116F?doi=10.1.1.79.5793, Sept. 2021.

[47] Qureshi M K. CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping. In *Proc. the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, Oct. 2018, pp.775-787. DOI: 10.1109/MICRO.2018.00068.

[48] Purnal A, Giner L, Gruss D *et al.* Systematic analysis of randomization-based protected cache architectures. In *Proc. the 42nd IEEE Symposium on Security and Privacy*, May 2021, pp.987-1002. DOI: 10.1109/SP40001.2021.00011.

[49] Bodduna R, Ganesan V, SLPSK P *et al.* Brutus: Refuting the security claims of the cache timing randomization countermeasure proposed in CEASER. *IEEE Computer Architecture Letters*, 2020, 19(1): 9-12. DOI: 10.1109/LCA.2020.2964212.



**Lu-Tan Zhao** received his B.E. degree in electronic information science and technology from Henan Polytechnic University, Zhengzhou, in 2014, and his M.E. degree in circuits and systems from University of Electronic Science and Technology of China, Chengdu, in 2017. He is a Ph.D. candidate at the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing. His current research interests include computer architecture and hardware security.

**Rui Hou** received his B.S. and M.S. degrees in computer architecture from Harbin Institute of Technology, Harbin, in 2001 and 2003 respectively, and his Ph.D. degree in computer architecture from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2007. He is a professor and vice director of State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing. He published over 30 papers in international conferences and journals, and got more than 50 patents. He has also served more than 10 times on conference organizing committees in international academia community. His current research interests include computer architecture, processor security, data center server architecture and AI security.

**Kai Wang** received his B.S. degree in software engineering from Harbin Institute of Technology, Harbin, in 2014, and his M.S. degree in computer science and technology from Harbin Institute of Technology, Harbin, in 2016. He is now a Ph.D. candidate at Harbin Institute of Technology, Harbin. His research interests include computer architecture and hardware security.

**Yu-Lan Su** received her Bachelor's degree in computer science and technology from Harbin Engineering University, Harbin, in 2019. She is now a Master student at the Institute of Information Engineering, Chinese Academy of Sciences, Beijing. Her current research interests include federated learning and cache side channel attacks.

**Pei-Nan Li** received his B.E. degree in software engineering from Shanxi University, Taiyuan, in 2014 and his M.E. degree in computer technology from Harbin University of Science and Technology, Harbin, co-educated with the Institute of Automation, Chinese Academy of Sciences, Beijing, in 2017. He is a Ph.D. candidate at the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing. His current research interests include computer architecture and hardware security.

**Dan Meng** received his Ph.D. degree in computer architecture from the Harbin Institute of Technology, Harbin, in 1995. He is the director of Institute of Information Engineering, Chinese Academy of Sciences, Beijing, and the dean of the School of Cyber Security, University of Chinese Academy of Sciences, Beijing. His research interests include high-performance computer architecture and cyber security.