Tail Victims in Termination Timing Channel Defenses beyond Cryptographic Kernels

Shijia Wei, Austin Harris, Yongye Zhu*, Prakash Ramrakhyani$^{\phi}$, Calvin Lin, and Mohit Tiwari

*The University of Texas at Austin, *University of California, Berkeley, $^{\phi}$ARM*

***Abstract*—Recent research in privacy-preserving systems relies on state-of-the-art defenses to close the termination timing channel. In this paper, we challenge the effectiveness of the adopted state-of-the-art defenses. In particular, we show that the two known practical defenses—predictive mitigation [5], [89] and fuzzy-clock [34], [49]—offer insufficient and unpredictably biased protections for general-purpose programs. To quantify this weakness, we introduce a new metric called "tail victim," which measures the fraction of secret inputs that unpredictably suffer the worst privacy leakage even after applying existing mitigations.**

To substantiate our findings, this paper first examines—for a range of privacy-sensitive applications, including web browsers, databases, and graph analytical programs—the leakage of input privacy through the termination timing channel. This paper then sheds further light on the leakiness of the state-of-the-art defenses for the termination timing channel, using the "tail victim" metric (1) to evaluate predictive mitigation and fuzzy-clock in above programs, (2) to evaluate real-world side-channel mitigations deployed in the Tor browser, and (3) to evaluate secure hardware designs that mitigate specific channels in isolation.

I. Introduction

The decades-old termination timing channel [39] leaks secret keys from cryptographic software through the amount of time used to execute cryptographic operations. Early work in side-channel defenses focused on defending the most sensitive workloads—namely, cryptographic kernels and keys—and this channel has been effectively closed for these cryptographic kernels, which have simple characteristics, such as short bit-length bounded loops, that make them easy to defend.

However, a wide variety of programs, including web browsers [52], ML classifiers [8], data storage [42], and serverless functions [67], require protections beyond just cryptographic keys. Unfortunately, existing termination timing channel elimination defenses for cryptographic software cannot be practically extended to general-purpose programs because these defenses restrict program behavior like network I/O, syscalls or paging (§ II-A).

As we move to defend privacy-sensitive applications, it is vital that we close this channel, because it is easily accessible to attackers from the entire hardware-software stack. For example, unprivileged adversaries can observe termination timing using contention on system services and on resources such as random number generators and virtual memory bookkeeping; they can use hardware contention on shared caches and bus bandwidth; and they can use physical signals such as power and electromagnetic radiation. Termination timing can also be measured remotely via network traffic or observed via external APIs. Moreover, in enclave or cloud environments, *privileged attackers manage scheduling, memory management, and resource access control, which can all be used to leak termination timings.*

Moreover, as a coarse-grained side channel, the termination timing channel is easy to exploit, requiring fewer measurements than fine-grained channels in microarchitectural resources like caches (§ IV-B).

*Yongye contributed to this research while at UT Austin.

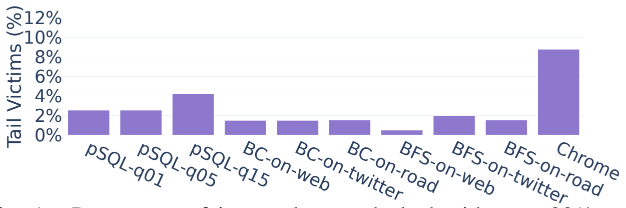


Fig. 1: Percentage of inputs that are leaked with over 90% top-3 accuracy even after applying predictive mitigation.

One single measurement is sufficient to initiate an attack. Repeated measurements can be leveraged with statistical or ML-based methods to improve accuracy and precision.

In this paper, we revisit the blindly relied upon mitigations for the termination timing channel. Using a broad class of *privacy-sensitive applications*, we show that the only *practical state-of-the-art* mitigations, predictive mitigation [5], [89] and fuzzy-clock [34], [49], which are commonly relied upon in privacy-preserving systems (elaborated in § II-A), break in unpredictable ways when used to defend a broader class of programs than cryptographic kernels (§ V). Here, "unpredictable" means that the defense breaks for certain inputs and certain programs, but it's difficult to know *a priori* when these breakdowns will occur. To quantify this weakness, we introduce the *tail-victim* metric (§ III), which measures the faction of victims that suffer the worst leakage in terms of input indistinguishability.

For example, Fig. 1 shows the percentage of tail victims that are still leaked with over 90% top-3 accuracy in a fingerprinting attack, even after applying predictive mitigation to our tested privacy-sensitive applications. For instance, in Chrome, close to 9% of our tested websites are still leaked with over 90% accuracy. Even worse, although predictive mitigation reduces input indistinguishability on average, its protections are distributed nonuniformly across the secret space, so some inputs become *more* distinguishable than without predictive mitigation, and it's difficult to predict which inputs will be affected negatively by predictive mitigation. Similarly, the protection from fuzzy-clock is inconsistent across programs: The randomness that works for one program may not sufficiently hide private inputs for others, and for some inputs the randomness may unnecessarily increase overhead.

We provide a detailed evaluation of predictive mitigation [5], [89] and fuzzy-clock [34], [49] in § V. We show that as we move away from defenses of cryptographic kernels, predictive mitigation *amplifies* leaks for inputs that are close to the time-boundary where the defense doubles its guess for the execution time (scheme described in V-A). On average, in our browser setup, predictive mitigation reduces attacker accuracy to 10.7% top-3, but the 93%tile tail-victim is classified with 100% accuracy for top-3 candidates. Similar to tail-latency, the tail-victim metric measures the worst-case leakage on a per-input basis. Across programs, on average 2.51% of the secrets are left unprotected with over 90% top-3 accuracy. We also show that fuzzy-clock works inconsistently. For example, a fuzzy-clock scheme fitted for a graph analytics program can reduce the average of attacker top-3 accuracy from 56.5% to 17%, but when the same scheme is applied to a browser, the average top-3 accuracy is as high as 41.5%. Different programs and inputs exhibit distinct multimodal timing distributions, so noise is insufficient to deceive attackers. Furthermore, we use the Tor browser with to show that the combination of predictive mitigation and the fuzzy-clock defense are insufficient in addressing the challenge exposed by the tail-victim metric: For fingerprinting victim websites, P95 top-3 classification accuracy is still 46% and P99 top-3 accuracy is over 80%.

This paper also sheds light on the implications of the termination timing channel on secure hardware designs. While adversaries can freely choose from among numerous microarchitectural resources to mount side-channel attacks, most solutions treat the termination timing channel as out of scope. However, doing so can leave a security gap *for any threat model that allows attackers to measure the termination timings*. We show that despite significant effort, redesigns of structures such as caches [60], [61], [83] or memory controllers [23], [24], [48], which aim to close various side channels for general-purpose programs, are still vulnerable to the termination timing channel (§ VI).

Specifically, we simulate betweenness centrality analysis of the USAroad location graph using the cycle-accurate full-system Gem5 simulator. An attacker confronted with a secure randomized cache cache [83] still achieves a 72.9% top-3 input-distinguishability (§ III); if we instead use a static-rate Path Oblivious-RAM [24], the attacker achieves a 95.4% top-3 input-distinguishability, with P50 tail-victims classified at nearly 100% (Fig. 16c). By contrast, our simulated baseline without side-channel resilient hardware shows only 67.8% top-3 input-distinguishability. Thus, architects need to consider the findings in this paper *before* placing point defenses against side-channel attacks into the microarchitecture. Otherwise, we could end up with a CPU with constant-time floating-point arithmetic, randomized and partitioned caches, and ORAM memory controllers, all with perhaps 100× throughput cost, *yet still* leak users' private inputs. For instance, ORAM-defense Phantom [48] aims to provide input privacy, but without termination timing channel protection, it does not provide input-indistinguishability for broader applications.

In summary, this paper makes the following contributions:

- We show that two state-of-the-art methods, predictive mitigation [5], [89] and fuzzy-clock [34], [49], which much prior research builds upon to close the termination timing channel, are unpredictably broken for broader applications (§ V) because they neglect the unpredictable timing distribution of programs and inputs.
- We introduce the *tail victim* metric (§ III), which quantifies the resulting privacy leakage of unreliable defenses. Tail victim measures the fraction of victims that suffer the worst leakage according to the input-distinguishability.
- We substantiate these findings using a set of real-world privacy-sensitive programs, including the PostgreSQL database, the GAP graph analytical programs, the Chrome browser, and the privacy-focused Tor browser.
- In addition to revealing the privacy impact that the termination timing channel has on broader applications, we reveal the implications of the termination timing channel for the secure side-channel resilient hardware (§ VI).

II. Background and Related Work

Program execution time leaks secrets as often demonstrated on cryptographic kernels [4], [11], [28], [39]. Termination time is defined as the length of an observable trace of program execution. For example, termination time can be determined by the dynamic instruction count or wall-clock time. As an *observation channel*, similar to side and covert channels in networking, an attacker in the termination timing channel passively observes a victim's behavior.

In this section, we first discuss the termination timing channel defenses and survey the literature of integrating such defenses in secure systems. Then, we discuss the related work in evaluating side-channel defenses like predictive mitigation.

A. Termination Timing Channel Defenses

Abundant efforts are spent in cryptography libraries to eliminate secret-dependent timing. These libraries ensure constant-time operations on secret keys, both by improving the algorithm [12], [16], [38], [41], [59] and by utilizing constant-time hardware [30], [47]. Researchers also proposed code-transformation techniques [9], [84] to make observable timing secret-independent for non constant-time programs. However, both Constantine [9] and SC-Eliminator [84] are designed for cryptographic software. SC-Eliminator relies on crypto-specific characteristics (e.g., bit-length bounded loops) to secure secret-dependent loops. Complex application behavior, such as secret-dependent syscalls and memory-mapped I/O create pathological scenarios for both. Moreover, compiler-enforced constant-time invariant can be broken by published or undocumented hardware optimizations (e.g., hardware store elimination).

Padding is another important primitive used in many defenses. Most notably, padding to worst-case execution time (WCET), in theory, closes the termination-timing channel. However, for broader applications, WCET results in tremendous overhead (§ IV-C). Execution-Leases [77] explored hardware-enforced per-function WCET, which outperforms end-to-end WCET. However, both hardware- [77] and software-enforced [10] WCET are evaluated using only short/cryptographic kernels, and greatly restrict protected program behavior. For instance, they prohibit syscalls, blocking I/Os, and paging. Meanwhile, bounded padding enjoys better performance by forceful early termination. Haeberlen *et al.* [31] proposed *Fuzz*, using bounded padding, tailored for differential-private database queries. Despite trading off query accuracy and utility, the bounded padding defense could incur $2.5-6\times$ overhead. This tradeoff between utility and privacy, however, is prohibitive for applications that rely on correct and complete executions.

Some mitigations trade off security for better performance and generalizability. Techniques such as predictive mitigation [5], [89] address the termination timing channel by predicting an execution time and exponentially doubling the padded execution time if the program doesn't terminate by the predicted time. fuzzy-clock techniques [34], [49] add noise to obfuscate the termination timing channel through randomly delaying the external observable termination event.

Predictive mitigation and fuzzy-clock remain the only applicable state-of-the-art defenses today for privacy-sensitive applications beyond cryptographic kernels, since both code transformation and WCET padding techniques fail to handle complex application behaviors like paging, I/O, and other syscalls.

B. Integrating Termination Timing Channel Defenses

To *close* the termination timing channel in secure systems for general-purpose programs, some designs employed WCET [35], [36], with worst-case performance. Many others choose to *mitigate* this channel. For example, predictive mitigation is widely employed in software-based solutions [7], [14], [29], [37], [54], [55], [62], [64], [72], [73], [76], [85], [90]. In the hardware context, an Instruction Set Architecture was designed to facilitate predictive mitigation [88], and two ORAM designs [24], [32] employed predictive mitigation.

Nonetheless, in our limited survey, at least 40 research papers acknowledge but disregard the termination timing channel. These studies target various threats of a secure system and generally consider the termination timing channel as an orthogonal issue, under the assumption that predictive mitigation has effectively closed this channel. However, we debunk this assumption in § V.

Specifically, six papers [2], [50], [57], [71], [94], [95] focusing on enabling privacy-preserving applications using hardware enclaves opt to exclude the termination timing channel from their scope. Similarly, six language and software runtime research focusing on using information flow security to defend against side-channel attacks [3], [58], [66], [78]–[80] also deem this channel out of scope.

In the hardware context, while numerous attacks [75] have been demonstrated on resources like memory bandwidth, address trace, caches, TLBs, functional units, and predictors, architects have focused mitigations on each side channel in isolation. That is, side-channel-resilient hardware designs are proposed for each vulnerable component. Therefore, it is not hard to find that recent defenses designed for closing specific microarchitectural side channels [1], [2], [6], [21], [22], [25], [45], [48], [63], [82], [94] acknowledge but ignore the termination timing channel.

To illustrate the importance of considering the termination timing channel, our work investigates similar *observation channels* in the microarchitecture. We discuss the last-level cache (LLC) and memory address trace side channels, which have drawn significant attention. To thwart such channels, side-channel-resilient designs seek to shape or obfuscate the observable trace. Some defenses add *noise* [96] or "fuzz" clocks [34], [40], [49] to the attacker's observations. Unfortunately, averaging or filtering techniques removes such noise with repeated observations [11], [65]. To obfuscate the observable traces, some designs leverage cryptographic randomness [46], [61], [83] to hide the victim access patterns. To combat the memory address trace side channel, Oblivious-RAM (ORAM) [26], [27], [74] with cryptographic guarantees that the trace is indistinguishable from a random trace, has been used in many hardware defenses [23], [24], [44], [48].

C. Related Work

Past work discussed in § II-A and II-B studied the average or upper bound of leakage in the termination timing channel. We raise the privacy issue that state-of-the-art solutions (e.g., predictive mitigation) unfairly and unpredictably protects inputs and introduce the tail-victim metric to quantify such weakness.

Limitation of predictive mitigation. Dantas *et al.* [18] observe that, for cryptographic kernels, non-deterministic termination event delivery results in a broader distribution of observable timings of the same input. This paper quantifies leakage (§ IV) in privacy-sensitive programs beyond cryptographic kernels. We also reveal a fundamental limitation (§ V) that the unmitigated termination timing of those programs may unpredictably fall across epoch boundaries in predictive mitigation, leaving a range of *tail victims* not protected.

Side-Channel metrics. Side-channel defenses are often measured with classic metrics such as mutual information or leakage bit-per-second. Several newly proposed metrics also advance the state of side-channel evaluation. They include probabilistic information flow graph (PIFG) [33] from He *et al.*, Side-Channel Vulnerability Factor (SVF) [19], [20] by Demme *et al.*, and Cache Side-channel Vulnerability (CSV) metric [91] by Zhang *et al.*. Overdorf *et al.* [53] investigate the fingerprintability as a metric for onion services under the network traffic side channel. Similar to the input-distinguishability proposed in this paper, they also criticize the use of aggregated metrics in evaluating side-channel defenses, and advocate a per-service analysis for defenses.

III. Methodology

To analyze the threat of the termination timing channel for general-purpose programs, and the limitations of the state-of-the-art defenses, we perform secret-input fingerprinting attacks on these programs. The goal of an attacker is to learn the class of the inputs that the program is executing on. In such an attack, adversaries profile offline a targeted subset of inputs and guess the victim secrets online. Statistical and ML methods have been explored (e.g. [81]) to improve fingerprinting in open-world settings, where victims may supply non-profiled inputs. In all experiments, we profile 25 timing samples for offline training, and use a separate set of 8 samples to test the online attack. The attacker uses Kolmogorov-Smirnov test for comparing distribution similarity to guess secrets. Candidates are sorted based on the probability of matching the online measured distribution to each offline profiled distribution.

Privacy-Sensitive suite: We study several privacy-sensitive general-purpose programs, such as a browser, a database, and graph algorithms, each with respective confidentiality property. We also include the AES and RSA crypto kernels as baseline.

For this test suite, we characterize the confidential inputs as follows: For browsers, the attacker's goal is to fingerprint the website origins visited by the victim. We use product shipping dates, volume, and revenue in TPC-H. Such secrets are often used in supporting proprietary business decisions. Here, the attacker is interested in learning the sensitive query inputs to the database. GAP programs perform analytics on graphs that represent social identities, geographical-locations, and web origins. The attacker's goal is to identify personal information, road locations, and website ownership, respectively.

Normalized Mutual Information (NMI): We use NMI [17] to estimate the leakage of secret keys in crypto kernel implementations from termination timing channel. Mutual information is a commonly-used information-theoretic measure that describes the number of bits about one variable may be learned from knowing the other. NMI, a relative metric between 0 and 1, normalizes the mutual information between variables $\mathscr{X}$ and $\mathscr{Y}$ towards the sum of the their entropy $H(\mathscr{X})$ and $H(\mathscr{Y})$. In side-channel context, mutual information has been used as a measure on the "amount of information" that traces $\mathscr{Y}$ may provide about secrets $\mathscr{X}$.

Tail victims and input-distinguishability: For general-purpose programs, we define input-distinguishability as how accurately an attacker can distinguish secret inputs. We use *top-k input-distinguishability*, where the secret input lies in an attacker's first k guesses, i.e., a top-1 input-distinguishability of 60% thus means the first guess is correct 60% of the time. This metric increases monotonically as an attacker makes more guesses (i.e., from top-1 to top-9), and can be represented as a stacked bar-graph. Furthermore, we introduce the *tail victim* metric by sorting the private victims based on input-distinguishability, and measure the fraction of victims that suffers the worst leakage. For example, P90tile tail victim reports the input-distinguishability of a victim who suffer a leakage larger than at least 90% of the inputs. Therefore, the term *tail victims* refers to the inputs that are least protected.

Experimental setup: In § IV and V, we evaluate the leakage of cryptographic and general-purpose programs. Specifically, we use Chrome (v85) with AlexaTop200 websites, PostgreSQL database (v12.1) with TPC-H (Q1, Q5, Q15), and the GAP benchmark suite (commit 4930a7)—betweenness centrality (BC), and breadth first search (BFS)—with real-world graphs from the suite (Web, Twitter, USARoad). For TPC-H, 120 inputs are uniformly selected from the input space, while 200 nodes are randomly selected as privacy sensitive nodes among millions in the GAP graph nodes. In addition, we use the AES-CBC and RSA-PKCS1 implementation in the OpenSSL library (v1.1.1e) with 200×200 random keys (pairs) and plaintext. Experiments are run in an Ubuntu 18.04 container on a Xeon E5-2620 v4 CPU. We assume attackers can only measure the wall-clock time instead of fine-grained timer like dynamic instruction or cycle count.

IV. Termination Timing Channel Motivation

A. Threat Model

In the termination timing channel, the attacker goal is to identify the secret inputs to the victim application. A secret input can be the bit-string of a program argument, or be implicit identities. For example, attackers may learn the victim identity in a social-media graph analytic program.

Attackers have multiple means for measuring termination timings. A local attacker can observe via OS resource contention, bursts of network traffic [13], microarchitecture utilization [93], or by physically observing the off-chip bus traffic [42] or device power [43]. The termination timing channel may also be exploited from a network attacker that (e.g.) controls a peer node/router or has access to the victim API. Moreover, *privileged attackers have direct visibility into victims' execution/termination status.*

In all these cases, the attacker can skip analyzing fine-grained execution traces, and use merely the length of the trace—this makes the termination timing an extremely accessible attack vector that is hard to plug via obfuscation techniques on network or cache traces.

Our attacker assumption is similar to [15], [43], [51], [52], [68]–[70], [87] where we assume the attacker can either co-locate with the victims or the attacker is hosting the service, or the attacker is a client of the same service as the victims. For example, Software-as-a-Service platforms like remote browsers, cloud-hosted databases, and image classification services give attackers access to the same or similar hardware platforms as victims. Specifically, for our browser experiments, we assume the victim visits an adversary-controlled website capable of measuring cache contention or querying the browser Network Information API to infer the sensitive page loading time. For databases, we assume an on-path network adversary can sniff traffic bursts in requests to learn the execution time of each query. For graph programs, we assume the attacker is the service provider who learns termination timing through collected telemetric traces. We collect the measurements in a system with only the attacker and victim software. System load may affect power, cache or interrupt-based fingerprinting attacks. Shusterman *et al.* [68]–[70] conducted extensive studies across different operating systems and hardware configurations to show the impact of such system factors.

B. Easily Exploitable

Compared to other fine-grained side channels, the termination timing channel is powerful as it requires far fewer measurements. We compare to two side channels that require fine-grained micro-architectural observations of the program executions, namely the cache occupancy [70] and the memory bandwidth [96] side channels.

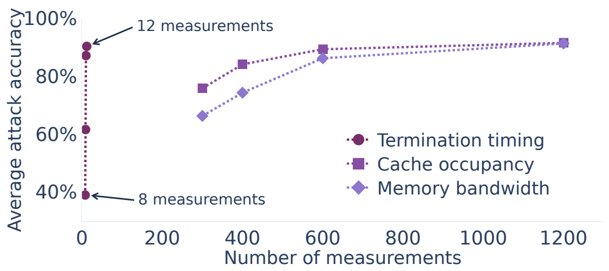
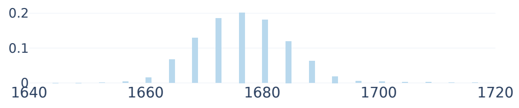
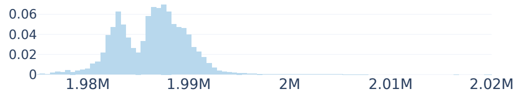


Fig. 2: Average top-1 accuracy in fingerprinting AlexaTop200 sites using three different side channels. Attacks that use the termination timing channel achieve comparable accuracy with 25× to 100× fewer observations.



(a) AES decryption (in **cycles**)



(b) RSA decryption (in **cycles**)

Fig. 3: Timing distribution of cryptographic kernels.

Cache occupancy and memory bandwidth traces are collected using hardware performance counters to provide accurate observations, which favors these two side channels in the evaluation.

Fig. 2 compares the three side channels in fingerprinting Alexa-Top200 websites. The y-axis shows the top-1 fingerprinting accuracy achieved by the attacker while the x-axis plots the number of measurements used in each attack. The termination timing channel achieves comparable accuracy with 25-100× fewer measurements than attacks using cache occupancy or memory bandwidth side channels. To exceed 90% accuracy, the termination timing channel requires 12 measurements while the other two use more than 1000 observations.

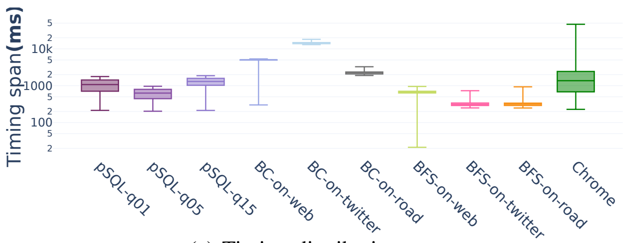
C. Input Privacy Leakage

Despite being easily accessible, the termination timing channel has been effectively closed for cryptographic software (§ II-A). However, the disparity in program behavior between cryptographic kernels and general-purpose programs leaves the question of whether protections extend beyond cryptographic software still open.

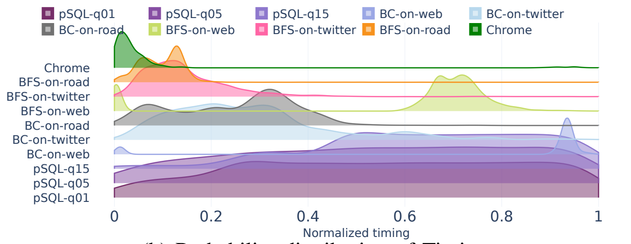
We start by showing that this channel leaks information in broader applications, whereas the count of observations in cryptographic software has been carefully engineered to be secret-independent. Then we examine the limitations of the mere two state-of-the-art mitigations in § V.

Cryptographic Kernels. To more accurately estimate the potential leakage in cryptographic kernel implementations from the termination timing channel, we measure the timings using high-resolution hardware time stamp counter.

AES and RSA kernels have way shorter termination times (less than 1*ms* in our setup) and narrower ranges compared to general-purpose programs. Fig. 3 shows the timing distribution in *cycles* for AES and RSA with different keys and ciphertexts. Meanwhile, Fig. 4 shows the timing distributions for general-purpose programs are often multimodal or with long tails.



(a) Timing distribution span



(b) Probability distribution of Timings

Fig. 4: Timing distribution of different inputs in various programs. Notably, cryptographic kernels (Fig. 3) operate at an extremely ephemeral time scale, while general-purpose programs span across larger scales and various distributions.

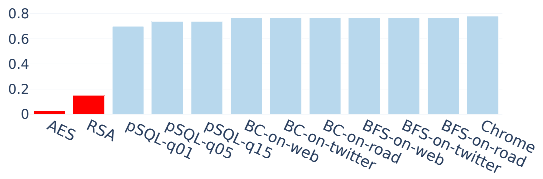
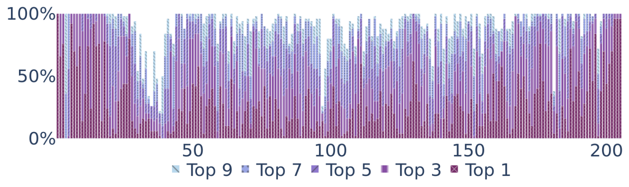


Fig. 5: Normalized Mutual Information (NMI) between secret inputs and termination timing for different programs.

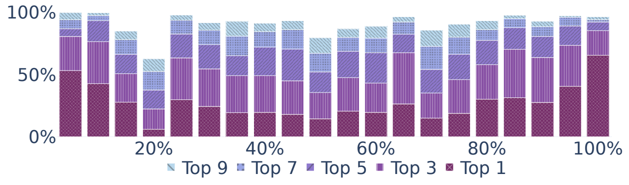
The NMI of AES and RSA in our experiments are 0.0266 and 0.148, respectively. This shows that the information available from termination timing channel is low for cryptographic kernels. For comparison, the NMIs of general-purpose programs are significantly higher, indicating a larger probability of leakage. Fig. 5 visualizes the difference. RSA shows slightly higher NMI than AES because the OpenSSL builtin key blinding scheme periodically updates the blinding seed and disturbs the timing for some program executions. However this variation is secret key independent.

General-Purpose Programs. To model more realistic attackers, we use a much coarser clock (micro-second) than the clock used for cryptographic kernels. First, we visualize the timing distribution in Fig. 4. In contrast to Fig. 3, these programs exhibit a wide range of variations in termination timing across inputs. This enables attackers to distinguish sensitive inputs from one another by merely observing the termination timing channel. We use *input-distinguishability* (defined in § III) as a metric of program vulnerability under the termination timing channel fingerprinting attack.

The results show that termination timing channel liberally leaks information about sensitive program inputs across all the general-purpose programs we tested. Specifically, Fig. 6 shows input-distinguishability for identifying the website origins being loaded in the Chrome web browser. X-axes in both subfigures are sorted based on the original median termination timing of loading each website. Fig. 6a shows detailed per-input distinguishability, while Fig. 6b provides a clustered view (averaged within each cluster) for a better readability. Fig. 6 includes top-1 to top-9 input-distinguishability, as stacked bars, colored from dark to light. Thus a darker color indicates a leakier input-distinguishability. On average, the attacker achieves 39.0% in top-1 and 86.8% in top-5 accuracy in identifying the origins among 200 websites. A close inspection on Fig. 6a reveals that overall the termination timing channel fingerprints the web origins in the Chrome with high accuracy. Websites 30 to 41 in Fig. 6a show low input-distinguishability compared to the rest of the inputs in the plot. This is because these websites are the same Google front page on different top-level domains (TLDs). These websites render extremely similarly or even display the same content which leads to similar termination timing distributions, and consequently lower input-distinguishability.



(a) Detailed per-input view.



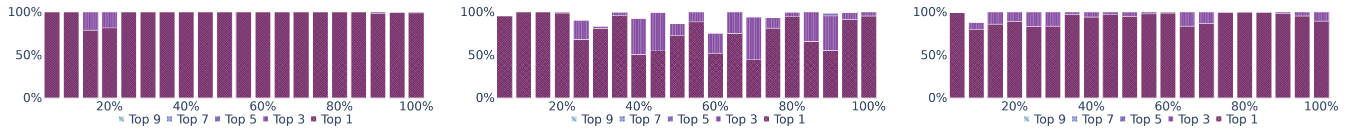
(b) Clustered view.

Fig. 6: Input-distinguishability across 200 different websites from top-1 to 9 guess. X-axis is sorted based on the original termination timing of loading each website in Chrome. From top-1 to top-9, the coloring goes from dark to light. Thus a darker plot indicates an overall leakier input-distinguishability.

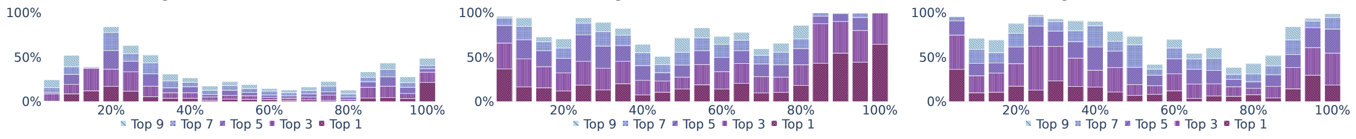
Fig. 7 show that besides leaking web origins in Chrome, the termination timing channel also leaks sensitive database query inputs like sales revenue, as well as input vertices (e.g. social identity) in the graph analytics programs. Fig. 7 plot the clustered view of input-distinguishability for all programs and their sensitive inputs. X-axes are sorted based on the median termination timing of each input.

Fig. 8 summarizes the average input-distinguishability for each program. In addition to Figs. 6 and 7, these results show that the termination timing channel is leaky: Figs. 7a to 7c show that for 120 query input values, an attacker can classify a query-serving database with 90.3% top-1 accuracy and 99.1% top-2 accuracy on average. Here input-distinguishability is very high because the termination timing is almost uniformly distributed over time depending on the total size of the data accessed by the database. For graph analytics programs on graphs with millions of nodes, an attacker can identify 200 random input nodes with confidence/accuracy ranging from 8% to 56.5% top-3, or 20.5% to 81.0% top-7.

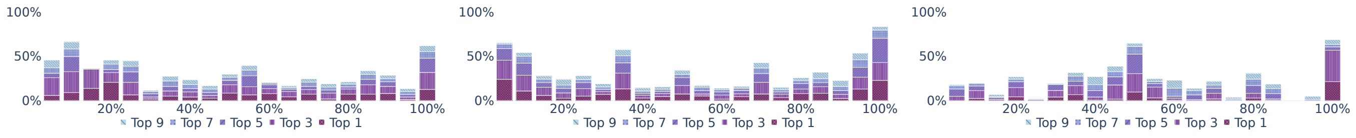
With Figs. 7d to 7f and Figs. 4a and 4b, we see that the termination timing channel leakage varies with the timing distributions of the secret inputs. Specifically, for program Betweenness Centrality (BC), termination timing channel leaks on the Twitter social graph and USARoad graph, while input-distinguishability on the ".sk domain web" graph is low with BC. Comparing Figs. 7e and 7h shows that input-distinguishability from the termination timing channel varies across the programs too. Specifically, with Breadth First Search (BFS), top-5 input-distinguishability is generally lower than 20%.



(a) PostgreSQL on TPC-H Q1 (b) PostgreSQL on TPC-H Q5 (c) PostgreSQL on TPC-H Q15



(d) BC on .sk domain webs (e) BC on Twitter social network (f) BC on USARoad graph



(g) BFS on .sk domain webs (h) BFS on Twitter social network (i) BFS on USARoad graph

Fig. 7: Input-distinguishability in the termination timing channel on various programs and their sensitive inputs.

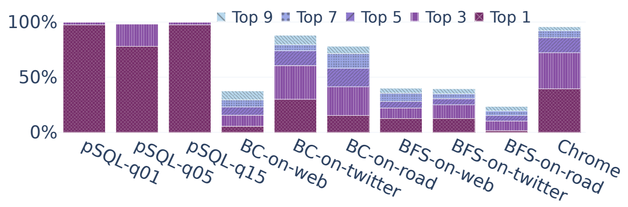


Fig. 8: Average input-distinguishability under an attacker with 8 timing samples per input for all programs.

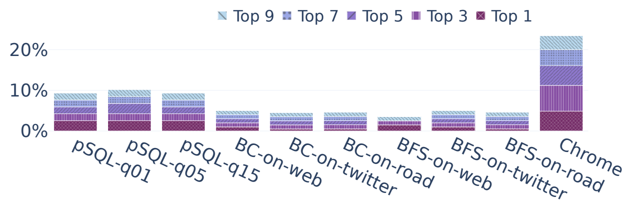


Fig. 9: Average input-distinguishability for applications with predictive mitigation. Predictive mitigation seemingly reduces the average leakage for all programs, resulting an overall input-distinguishability lower than 10%.

However, with BC, an attacker is able to achieve over 70% top-5 accuracy from the termination timing channel. This difference is from the BC implementation in GAP leveraging heavily data-dependent optimizations to obtain a fast approximation of the graph betweenness centrality. For BFS, although it exhibits data-dependent access patterns, it traverses down the entire graph.

One temptation left though is to leak *only* a few bits, which we show next has its pitfalls.

V. State-of-The-Art Solutions are Broken

In this section, we show that two mitigations for the termination timing channel namely, predictive mitigation [5], [89] and fuzzy-clock [34], [49] break when protecting a broad class of privacy-sensitive applications. Both aim to provide a tradeoff between security and performance for broader applications, by leaking a few bits to improve performance. Predictive mitigation enforces determinism in termination time while fuzzy-clock leverages randomness. We assume that the attacker is able to profile the termination time of mitigated programs across different inputs. As with our analysis in the previous section (§ IV), we measure the attackers' ability to distinguish among a set of plausible inputs of attackers' interest based on the perturbed termination time.

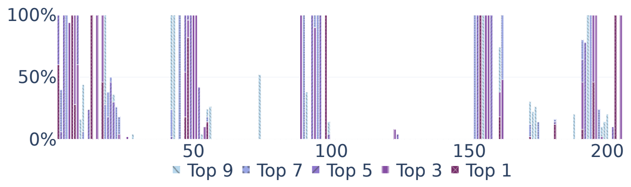


Fig. 10: Per-Input-distinguishability (on the y-axis) across different website inputs *after applying predictive mitigation to Chrome*. The X-axis is the web origins sorted as in Fig. 6. Thus, for instance, website 38 can be distinguished by an attacker with almost 100% accuracy.

A. Predictive Mitigation

Predictive mitigation [5], [89] aims to bound the amount of information leakage by bucketing program execution time into epochs. The length in each epoch is twice that of the previous epoch: when program termination misses the current epoch deadline, predictive mitigation moves to the next epoch, doubling the predicted termination timing. The total execution time is padded to the smallest enclosing epoch boundary. Predictive mitigation is the best known practical termination timing channel solution for general privacy-sensitive applications, which recent security research relies on. In particular, 6 recent papers [2], [50], [57], [71], [94], [95] root their side-channel security assumptions in the effectiveness of predictive mitigation to enable privacy-preserving applications using hardware enclaves. Predictive mitigation in theory bounds the average leakage; however, our experiments next show that it provides only biased protection and practically *amplifies* the leakage for certain inputs.

Indeed, the *average* input-distinguishability achieved by predictive mitigation, as depicted in Fig. 9, is fairly low compared to the unmitigated case (Fig. 8). For example, the average top-3 input-distinguishability for the Chrome browser drops to only 10.7% from 69.8%. However, this average metric is deceptive. When closely examined, Fig. 10 reveals that the benefit of reduced leakage is

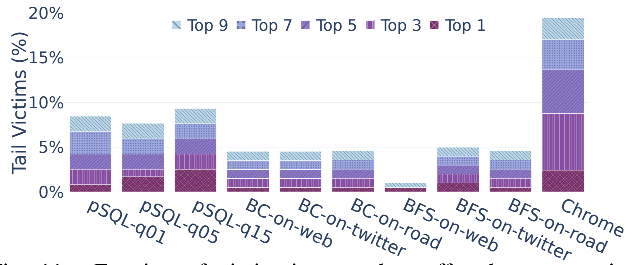


Fig. 11: Fraction of victim inputs who suffer the worst privacy leakage over 90% accuracy after applying the predictive mitigation.

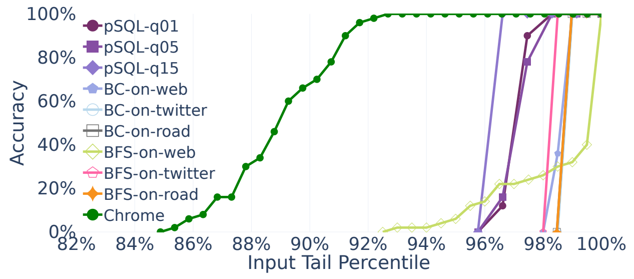


Fig. 12: Tail input-distinguishability (top-3) after predictive mitigation against an attacker with only 8 timing samples. More than 7% of Chrome's inputs are still perfectly leaked.

not uniformly distributed among victim inputs. Fig. 10 plots the per-input distinguishability for the Chrome browser after applying predictive mitigation. The X-axis is sorted based on the original termination timing for loading these websites in Chrome. The spikes in Fig. 10 correspond to inputs that predictive mitigation fails to protect (thus still have a high distinguishability). We refer to these victim inputs as tail victims, who suffer the worst privacy leakage—analogous to tail latency in the field of web application benchmarking. We find that the tail victims form clusters on the X-axis. The clustering behavior suggests that these tail victim inputs have original termination timings that are within a similar range, given that the X-axis is sorted based on the original termination timing of the inputs. More concretely, we find that predictive mitigation fails to protect these inputs whose termination times distribute across the epoch-boundaries where predictive mitigation doubles the length of the next epoch for the program's termination time.

Besides Chrome, we find there exist similar tail victims among other applications tested with predictive mitigation employed. Fig. 11 shows a stacked bar graph of the fraction of privacy-sensitive inputs who suffer an input-distinguishability greater than 90% given different top-N metrics. This indicates that predictive mitigation fails to protect these tail victims across all our tested applications, with BFS-on-web being seemingly least impacted. However, Fig. 12 suggests even victim inputs in BFS-on-web are not safe from the biased protection of predictive mitigation. Fig. 12 plots the tail top-3 input-distinguishability. The y-axis shows the attacker accuracy in fingerprinting the sensitive inputs, whereas the X-axis sorts the inputs based on the probability (given in percentiles) each input can be correctly identified. Although the percentage of inputs at more than 90% top-3 input-distinguishability in BFS-on-web is small, Fig. 12 uncovers that BFS-on-web actually hosts a larger fraction of inputs that fall victim of the biased protection from predictive mitigation. Fig. 12 also shows that although predictive mitigation eliminates leakage from the termination timing channel for about 85% of Chrome's inputs, it leaves 9% of the inputs with more than 80% probability of being accurately fingerprinted, and 7% of the inputs unfortunately always leaked.

This inconsistency across sensitive inputs is due to the nondeterministic timing behavior in the system. The termination timings of each program input execution may vary from run-to-run due to system noise. This run-to-run variation shapes the termination timing distribution of each input into a near-normal distribution. After applying predictive mitigation, the resulting distribution for most inputs is simply unimodal. However, for some inputs, the run-to-run variation causes their original termination timing to span across the prediction boundaries. Predictive mitigation therefore transforms their termination timing distributions from near-normal distributions to bimodal (or even multimodal in extreme cases) distributions which has distinct peaks and can be of different densities. For these inputs, the attacker's task becomes differentiating skewed bimodal or multimodal distributions from unimodal distributions, which is statistically easier than differentiating two near-normal distributions. Therefore, by comparing Fig. 10 to Fig. 6a, we find that interestingly (though not surprisingly), predictive mitigation may actually exacerbate input-distinguishability. This occurs for the victim inputs whose termination time is distributed across epoch boundaries. For instance, after applying predictive mitigation, website 48 suffers an 88% top-1 input-distinguishability which was merely 16% before mitigation. Similarly, top-1 input-distinguishability for website 98, increased from 0 to 100%.

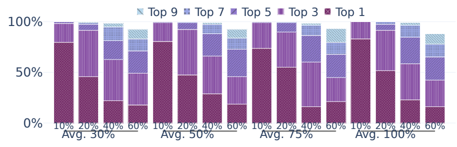
We conclude from Figs. 10 to 12 that **predictive mitigation *unpredictably* provides only skewed privacy protection across inputs, as many inputs can still be easily distinguished by an attacker.** Predictive mitigation unfairly sacrifices the privacy of certain inputs in return for a bounded average. The *tail* effect and the average metric in the predictive mitigation are particularly pernicious for end-user privacy. Since it is impossible to pre-determine the termination time of all possible privacy-sensitive inputs, it is impossible to know a priori which private inputs predictive mitigation will fail to protect unless exhaustive measurements are taken for executing all possible privacy-sensitive inputs.

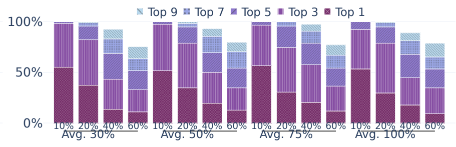
B. *Fuzzy-Clock*

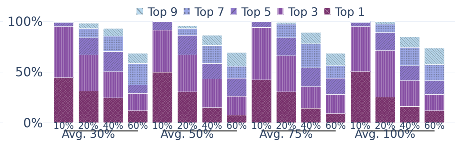
The fuzzy-clock idea may seem trivial: adding randomized noise—so that the defense avoids substantial overhead—makes the adversary's task harder. Many prior proposals omit exact details. In this section, we reveal the limitations of fuzzy-clock with a detailed analysis.

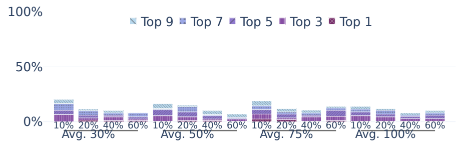
Noise can be added on a per-input basis, i.e. program executions of the same input will be randomized deterministically to a new termination time. For example, a hash-based randomization scheme will shape the executions based on the resulting hash. These schemes rely on churning the randomization to prevent the attacker from building a profile within the churn period. If the attacker manages to do so, these schemes provide no protection for the termination timing channel. Therefore in this section, we focus on per-execution noise rather than per-input.

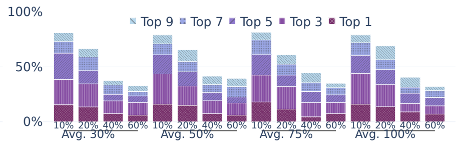
We vary the average noise as well as the noise spread. The noise attributes are presented as a ratio of each program's termination timing range. The average of the noise determines the average performance overhead, while the spread and average together determine the worst-cast overhead. Note that an average of 100% noise has *larger* performance overhead than padding all executions to the worst-case termination time. Figs. 13 and 14a shows the input-distinguishability


(a) PostgreSQL on TPC-H Q1


(b) PostgreSQL on TPC-H Q5


(c) PostgreSQL on TPC-H Q15


(d) BC on .sk domain webs


(e) BC on Twitter social network

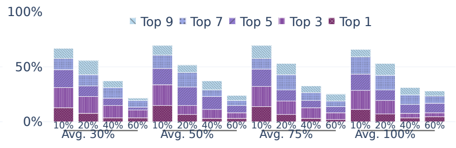
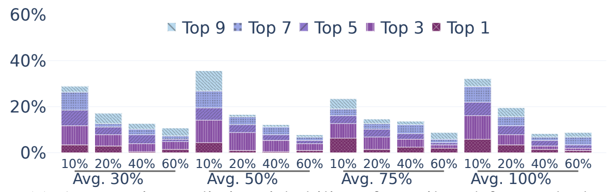

(f) BC on USARoad graph

Fig. 13: Average input-distinguishability after applying fuzzy-clock schemes with different average and spread attributes. We use a subset of test suite whose original top-3 input-distinguishability is higher than 40%. The noise attributes are presented as a ratio to each program's termination timing range. For example, an average 50% noise for a program that terminates between 100s and 1000s is 550s. An average of 100% noise has *larger* performance overhead than padding all executions to the worst-case termination time.


(a) Average input-distinguishability after tailored fuzzy-clock.

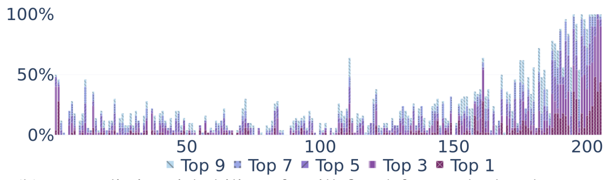

(b) Input-distinguishability after ill-fitted fuzzy-clock scheme.

Fig. 14: Chrome after applying (a): program-specific fuzzy-clock scheme, (b): fuzzy-clock scheme designed for BC-on-Twitter. The scheme in (b) lowers the average top-3 input-distinguishability for BC-on-Twitter to 17% while leaving Chrome with an top-3 average as high as 41.5%.

after applying fuzzy-clock schemes on the example programs (a subset of the test suite whose original top-3 input-distinguishability are larger than 40%). The X-axes in each subfigure show 4 groups, within each the spread of the noise increases while the average only increases across groups. Comparing across groups, the average noise size has little impact on the protection. However, increasing the noise spread offers better leakage reduction.

Different programs (Fig. 13a vs. 13d, 14a) and different data-sets (Fig. 13d vs. 13e, 13f) enjoy distinct protections using the same noise ratio. For example, with a noise average of 30% and 40% spread, postgreSQL is only mitigated to around 80% top-5 input-distinguishability, while 40% of noise spread lowers the top-5 average to less than 10% in Chrome. For BC to achieve a top-5 input-distinguishability lower than 30%, we need to add a 40% spread for the twitter graph, 20% for road, and less than 10% for the web graph, respectively.

Due to the strictly additive nature of timing noise, that is, it is impossible to add *negative* noise to time, an adversary could prune the list of potential secret information. Specifically, suppose a fuzzy-clock scheme adds noise ranging from 0 to N. Given a fuzzed termination time X, the attacker knows that the original termination timing T is such that $max(X-N,0) \leq T \leq X$. The *max* function is important because time cannot be negative. This provides important clues to the attacker to guess the actual termination time. When $X < N$, there are fewer plausible actual execution times compared to when $X \geq N$.

A defense that was based on unimodal leakage distribution is likely insufficient to hide all peaks in the programs that have multimodal distribution. Fig. 14 illustrates this inconsistency. With a fuzzy-clock scheme designed for BC on the Twitter graph, Chrome still has a 95%-tail top-1 input-distinguishability of 92% and an average top-3 at 41.5%. The scheme manages to hide inputs for BC on the Twitter graph to an average top-3 input-distinguishability as low as 17%. However, it fails to hide the two peaks in Chrome's termination timing distribution (shown in Fig. 4b). As a result, the rightmost 20 inputs—that fall into the long-termination time peak—in Fig. 14b exhibit near-perfect input-distinguishability.

Therefore, fuzzy-clock cannot be applied at the system level or transparently. The extent of noise needed to satisfactorily hide the inputs varies dramatically based on the application and its data. Indeed, fuzzy-clock defenses can add increasingly more noise to cover all the extreme peaks. However, doing so runs directly against the motivation of adding noise–trading off security for performance.

C. *Leakage in Secure Software: The Tor Browser*

Next, we investigate whether the real-world deployment of the fuzzy-clock mitigation in the Tor browser (v12) together with predictive mitigation could close the termination timing channel. This study shows how countermeasures in existing privacy-focused secure software help mitigate the termination timing channel, whether predictive mitigation improves user privacy in the Tor browser, and to what extent.

The Tor browser is a patched version of Firefox, which aims to provider its users better privacy by deploying anti-tracking and anti-fingerprinting techniques. Specifically, against timing-side-channel-based fingerprinting attacks, the Tor browser relies on clamping the explicit clock resolution to 100ms and adding jitters [56]. In this evaluation, we run a JavaScript-based attacker who measures the cache contention as a result from the Tor browser loading and rendering the websites, similar to our Chrome experiments. Instead of inspecting the contention traces and training a machine-learning classifier on these traces, the modeled attacker fingerprints websites rendered in the Tor browser with the termination timing channel. The attacker determines the termination timing for each collected trace using a common static threshold.

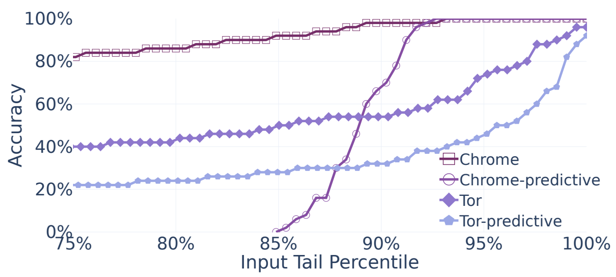
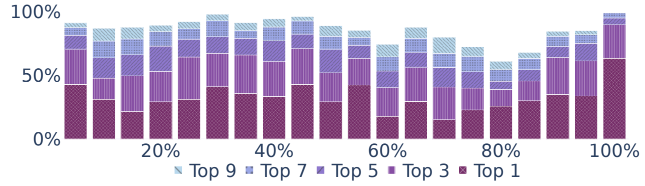


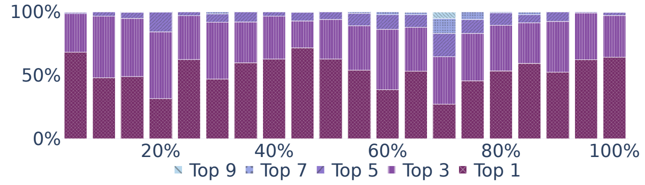
Fig. 15: Top-3 tail victims from fingerprinting websites in the Tor and Chrome browsers using the termination timing channel. "Tor-predictive" and "Chrome-predictive" represent browsers with predictive mitigation enabled.

System	OS	Redhat 8 with Linux kernel 5.4.49
	Processor	4 x86 OoO Cores at 3GHz
Core	Predictor	LTAGE and Indirect Predictor, 512-entry BTB
	Fetch	5 wide Fetch, Decode, Rename, 224-entry ROB
	Dispatch	8 wide Dispatch, Issue, Writeback, 97-entry IQ
	Exec	4 INT ALUs, 3 INT VectU, 2 FP FMAs, 168/180 Phys. Reg., 72/56-entry Ld/St Buffer
Memory	L1-I/D	32kB, 8-way, 2/4 cycles, 16-entry MSHR, LRU
	L2	256kB, 4-way, 10 cycles, 20-entry MSHR, LRU
	Shared L3	8MB, 40 cycles, 256-entry MSHR, stride prefetch, LRU or ScatterCache [83] with 16 partitions
	DRAM	8GB, 4 Channels, DDR4-2400, DRAMSim2 or static-rate 900-cycle ORAM [24]

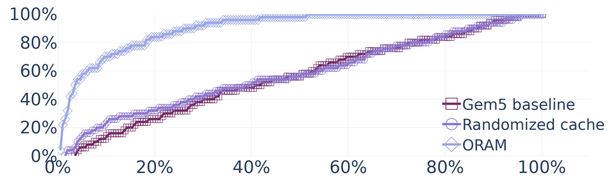
TABLE I: Skylake-like CPU in gem5 full-system simulation.



(a) With a 16-skew ScatterCache [83] LLC.



(b) With a static-rate Path ORAM (A secure baseline used in [24]).



(c) Top-3 tail victims in simulated systems.

Fig. 16: Input-distinguishability for BC on USARoad graph with simulated baseline and side-channel resilient hardware.

We compare the vanilla Tor browser with one that employs predictive mitigation. Fig. 15 plots the top-3 tail-victim metric for the two Tor browsers as well as results from the Chrome experiments for reference. Solid markers depict the tail victims for the Tor browsers while open markers show tail victims for the Chrome browsers. "Tor-predictive" and "Chrome-predictive" represent browsers with predictive mitigation enabled.

Comparing the vanilla Tor browser (diamond) with the baseline Chrome browser (open square), we see that for 75% of the inputs (i.e., websites), the fuzzy-clock mitigation employed in the Tor browser is able to keep top-3 attacker classification accuracy at 40% or lower, compared to an up-to 80% bound in the baseline Chrome. However, the fuzzy-clock mitigation does poorly for the rest 25% tail victims, some of whom suffer a top-3 classification accuracy as high as 95%. We also see that while predictive mitigation eliminated leakage from the termination timing channel for 85% of websites in the Chrome browser, (i.e., P85 is 0), it fails to achieve the same in the Tor browser at the 85%tile tail. Employing predictive mitigation in the Tor browser reduces the P95 (95%tile tail) top-3 accuracy from 74% to 46%. However, 1.4% of the websites are still left with over 80% classification accuracy.

VI. Implications on Secure Hardware Designs

In this section, we demonstrate that termination timing channel persists even when microarchitectural resources are designed to eliminate side channels. We perform the same fingerprinting attack as in § IV-C, under simulated systems using the cycle-accurate gem5 simulator. Table I details the baseline Skylake-like CPU configuration. We evaluate the termination timing channel leakage when the processor employs one of the two side-channel-resilient secure hardware: (1) a 16-skew ScatterCache [83] as the LLC; (2) a static-rate 900-cycle Path Oblivious-RAM memory controller used in [24] as a secure baseline. Due to the limit of gem5 full-system simulation speed, we evaluate the BC program on the USARoad graph.

Both secure hardware components obfuscate program execution traces. Randomized LLC designs obfuscate the cache access pattern to prevent LLC access trace-based fingerprinting. The evaluated ScatterCache leverages a skewed, pseudorandom cache indexing function, which uses keyed mapping between addresses and set indices. By producing one index for each cache way, ScatterCache randomizes the composition of cache sets, making observing access trace through cache conflict significantly harder. Oblivious-RAM memory controllers close the memory bus address-trace side channel by obfuscating the memory access addresses. The Path ORAM design manages the external memory as a binary tree. While each data block is mapped randomly to a leaf of the binary tree, they are stored on the path from the root of the tree to its mapped leaf. Obfuscation is achieved by accessing a data block along with the entire tree path of blocks and remapping data block to a different leaf after each block access. Additionally, the static-rate Path ORAM closes the memory timing and utilization channel.

Notably, Path ORAM is specifically designed for input privacy: The memory address traces of executing two identical inputs are made indistinguishable from traces of two distinct inputs, from a cryptographic perspective. *Unfortunately, we show that neither of the two secure hardware components improves input-distinguishability if attackers leverage the easily accessible termination timing channel.*

Fig. 16 shows the input-distinguishability on simulated termination times for the BC analytical program on the USARoad graph. The average top-3 accuracy is 72.9% (Fig. 16a) for ScatterCache and 95.4% (Fig. 16b) for ORAM. Both secure hardware designs result in higher input-distinguishability than simulated baseline hardware at 67.8% and a real-hardware baseline at 40.8% (Fig. 7f) without randomized LLC or ORAM[1]. The system with ORAM leaks more than the system with randomized LLC (Fig. 16b is darker than Fig. 16a). This is because a static-rate ORAM reduces memory

[1]It is expected that the simulated baseline system exhibits higher leakage than real hardware due to limited non-determinism in gem5.

level parallelism by serving one request at a time, thus making the termination timings across all inputs more distinct. However, randomized LLC obfuscates the cache access pattern without disturbing the parallelism in program executions significantly. Fig. 16c shows the attacker accuracy in identifying the top-3 tail victims in three simulated systems. The ORAM line rises above the baseline and randomized cache systems, with P20 tail-victim inputs being fingerprinted at above 80% accuracy. The similar trend between the randomized cache and the baseline indicates that the randomness introduced in randomized LLC barely shapes leakage distribution.

A key implication of this result is that if one threat model allows attackers to measure/learn the termination timings, programs might still leak substantially even with secure caches, memory controllers, etc. turned on. In such cases, a safer trade-off might be to not employ, for example, secure caches and instead opt for a dedicated machine operated with strictly controlled termination-channel defenses.

VII. Countermeasures and Conclusion

Direction and challenges for countermeasures. The fact that state-of-the-art solutions are unreliable for general-purpose programs leaves us with worst-case padding as the only solution today. If such overhead is unacceptable, isolated security cores with deterministic timing and dedicated single-tenant machines may be the only option for securing general privacy-sensitive applications for the time being.

The wide range of execution times across inputs, the non-uniform distribution of these timings, and the unrestricted privacy-dependent program behavior, suggest that cryptography-oriented defense techniques like constant-time execution may be impractical for a broad class of privacy-sensitive applications. Moreover, the protection inconsistency among different inputs and programs roots in the distinct, non-uniform distributions of the observable side-channel traces across all inputs and programs. This non-uniform distribution challenges the protection schemes that seek amortized security and dooms the privacy of those tail victims. As a result, a privacy-centric metric for input-distinguishability is required for the termination timing channel, similar to system level channels [86], [92]. Metrics like Differential Privacy or K-Anonymity, which provides a protection guarantee for each input, could be explored. However, closing the termination timing channel poses a crucial difference that *the noise is only one-sided*—we cannot reduce execution time.

Realizing the privacy-centric metrics for general-purpose programs also presents a unique cross-stack challenge: Not only can termination timings be measured across the entire stack, but the source of the leakage resides in both the application and the hardware. Data-dependent control flows, computations, and system calls in software, caching, predictions, coalescing in hardware, and even undocumented optimizations all contribute to the termination timing channel. This makes it impossible to effectively address this channel in any layer alone. Instead, cross-stack defenses, enabling applications to pass requirements to all system layers, are needed. As architects, we need to consider the termination timing channel *before* re-designing microarchitectural resources with point solutions because the termination timing channel is observable across many threat models and the leakage remains even if other channels are closed.

Concluding remarks. In this paper, we have demonstrated the weakness of the state-of-the-art mitigations, namely predictive mitigation and fuzzy-clock, for defending the termination timing channel in broad privacy-sensitive applications. These mitigations unpredictably bias protection among private inputs. Unfortunately, much recent research relies upon these defenses for building privacy-preserving systems. We have alsopaper introduced the *tail-victim* metric in § III. We have used this metric to quantify, for the first time, the privacy leakage in the state-of-the-art mitigations for privacy-sensitive applications. This metric also underlines the unpredictable protection distribution of the mitigations. Lastly, this paper has explored the tail-victim challenges in existing side-channel mitigations deployed in the Tor browser, as well as in future secure hardware proposed to mitigate side channels.

More broadly, this paper is a call to arms: we invite architects (1) to look beyond cryptographic primitives, (2) to design cross-stack solutions that help protect user inputs, especially tail victims, at a low cost.

Acknowledgments

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and by Intel Side Channel Academic Programming (SCAP) and Resilient Architectures and Robust Electronics (RARE) research centers. Shijia is also supported in part by the Qualcomm Innovation Fellowship 2022.

References

[1] S. Aga and S. Narayanasamy, "Invisimem: Smart memory defenses for memory bus side channel," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 2017.

[2] S. Aga and S. Narayanasamy, "InvisiPage: Oblivious demand paging for secure enclaves," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, 2019.

[3] A. Askarov, S. Chong, and H. Mantel, "Hybrid monitors for concurrent noninterference," in *2015 IEEE 28th Computer Security Foundations Symposium*. IEEE, 2015.

[4] A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands, "Termination-insensitive noninterference leaks more than just a bit," in *European symposium on research in computer security*. Springer, 2008.

[5] A. Askarov, D. Zhang, and A. C. Myers, "Predictive Black-box Mitigation of Timing Channels," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010.

[6] A. Azevedo de Amorim, N. Collins, A. DeHon, D. Demange, C. Hriţcu, D. Pichardie, B. C. Pierce, R. Pollack, and A. Tolmach, "A verified information-flow architecture," *Journal of Computer Security*, vol. 24, no. 6, 2016.

[7] M. Backes, G. Doychev, and B. Köpf, "Preventing side-channel leaks in web traffic: A formal approach," in *NDSS*, 2013.

[8] S. Banerjee, S. Wei, P. Ramrakhyani, and M. Tiwari, "Triton: Software-Defined Threat Model for Secure Multi-Tenant ML Inference Accelerators," in *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '23. New York, NY, USA: AMC, Oct. 2023.

[9] P. Borrello, D. C. D'Elia, L. Querzoni, and C. Giuffrida, "Constantine: Automatic side-channel resistance using efficient control and data flow linearization," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[10] B. A. Braun, S. Jana, and D. Boneh, "Robust and Efficient Elimination of Cache and Timing Side Channels," Aug. 2015.

[11] D. Brumley and D. Boneh, "Remote Timing Attacks Are Practical," in *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, ser. SSYM'03. Berkeley, CA, USA: USENIX Association, 2003.

[12] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*. Boston, MA: Springer US, 1983.

[13] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *2010 IEEE Symposium on Security and Privacy*, 2010.

[14] D. Cock, Q. Ge, T. Murray, and G. Heiser, "The last mile: An empirical study of timing channels on sel4," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[15] J. Cook, J. Drean, J. Behrens, and M. Yan, "There's always a bigger fish: A clarifying analysis of a machine-learning-assisted side-channel attack," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: AMC, Jun. 2022.

[16] J.-S. Coron, "Higher Order Masking of Look-Up Tables," in *Advances in Cryptology – EUROCRYPT 2014*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014.

[17] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[18] Y. G. Dantas, R. Gay, T. Hamann, H. Mantel, and J. Schickel, "An evaluation of bucketing in systems with non-deterministic timing behavior," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2018.

[19] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-channel vulnerability factor: A metric for measuring information leakage," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Portland, Oregon: IEEE Computer Society, Jun. 2012.

[20] J. Demme and S. Sethumadhavan, "Side-channel vulnerability metrics: Svf vs. csv," in *Proc. of 11th Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, 2014.

[21] P. W. Deutsch, Y. Yang, T. Bourgeat, J. Drean, J. S. Emer, and M. Yan, "DAGguise: Mitigating memory timing side channels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2022. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3503222.3507747

[22] A. Ferraiuolo, Y. Wang, R. Xu, D. Zhang, A. Myers, and E. Suh, "Full-processor timing channel protection with applications to secure hardware compartments," 2017.

[23] C. W. Fletcher, L. Ren, A. Kwon, M. v Dijk, E. Stefanov, D. Serpanos, and S. Devadas, "A Low-Latency, Low-Area Hardware Oblivious RAM Controller," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2015.

[24] C. W. Fletcher, L. Ren, X. Yu, M. van Dijk, O. Khan, and S. Devadas, "Suppressing the Oblivious RAM timing channel while making information leakage and program efficiency trade-offs," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium On*. IEEE, 2014.

[25] Q. Ge, Y. Yarom, F. Li, and G. Heiser, "Your processor leaks information-and there's nothing you can do about it," *arXiv preprint arXiv:1612.04474*, 2016.

[26] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Privacy-preserving group data access via stateless oblivious RAM simulation," in *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. SIAM, 2012.

[27] M. T. Goodrich, O. Ohrimenko, and R. Tamassia, "Data-oblivious graph drawing model and algorithms," *CoRR*, vol. abs/1209.0756, 2012.

[28] J. Großschädl, E. Oswald, D. Page, and M. Tunstall, "Side-channel Analysis of Cryptographic Software via Early-terminating Multiplications," in *Proceedings of the 12th International Conference on Information Security and Cryptology*, ser. ICISC'09. Berlin, Heidelberg: Springer-Verlag, 2010.

[29] P. Grubbs, T. Ristenpart, and Y. Yarom, "Modifying an enciphering scheme after deployment," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017.

[30] S. Gueron, "Intel Advanced Encryption Standard (AES) New Instructions Set," Intel, White Paper, May 2010.

[31] A. Haeberlen, B. C. Pierce, and A. Narayan, "Differential privacy under fire," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. USA: USENIX Association, Aug. 2011.

[32] S. K. Haider, O. Khan, and M. van Dijk, "Revisiting definitional foundations of oblivious ram for secure processor implementations," *arXiv preprint arXiv:1706.03852*, 2017.

[33] Z. He and R. B. Lee, "How Secure is Your Cache Against Side-channel Attacks?" in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17, ACM. New York, NY, USA: ACM, 2017.

[34] W.-M. Hu, "Reducing timing channels with fuzzy time," in *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1991.

[35] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel, "Telekine: Secure computing with cloud gpus," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020.

[36] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016.

[37] S. Kananizadeh and K. Kononenko, "Predictive mitigation of timing channels-threat defense for machine codes," *Journal of Grid Computing*, vol. 15, no. 3, 2017.

[38] E. Käsper and P. Schwabe, "Faster and Timing-Attack Resistant AES-GCM," in *Cryptographic Hardware and Embedded Systems - CHES 2009*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5747.

[39] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, 1996.

[40] D. Kohlbrenner and H. Shacham, "Trusted Browsers for Uncertain Times," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016.

[41] R. Könighofer, "A fast and cache-timing resistant implementation of the AES," in *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA'08. San Francisco, CA, USA: Springer-Verlag, Apr. 2008.

[42] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, Aug. 2020.

[43] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein, "Power to peep-all: Inference attacks by malicious batteries on mobile devices," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, 2018.

[44] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015.

[45] C. Liu, M. Hicks, and E. Shi, "Memory trace oblivious program execution," in *2013 IEEE 26th Computer Security Foundations Symposium*. IEEE, 2013.

[46] F. Liu and R. B. Lee, "Random Fill Cache Architecture," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Cambridge, United Kingdom: IEEE Computer Society, Dec. 2014.

[47] A. Ltd, "Arm® Architecture Registers Armv8, for Armv8-A architecture profile | DIT," https://developer.arm.com/docs/ddi0595/e/aarch64-system-registers/dit.

[48] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, "PHANTOM: Practical oblivious computation in a secure processor," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. Berlin, Germany: AMC, Nov. 2013.

[49] R. Martin, J. Demme, and S. Sethumadhavan, "TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *39th International Symposium on Computer Architecture (ISCA 2012), June 9-13, 2012, Portland, OR, USA*, 2012.

[50] A. Oak, A. M. Ahmadian, M. Balliu, and G. Salvaneschi, "Language support for secure software development with enclaves," in *IEEE Computer Security Foundations Symposium (CSF 2021)*, 2021.

[51] O. Ohrimenko, M. Costa, C. Fournet, C. Gkantsidis, M. Kohlweiss, and D. Sharma, "Observing and Preventing Leakage in MapReduce," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: AMC, Oct. 2015.

[52] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: AMC, Oct. 2015.

[53] R. Overdorf, M. Juarez, G. Acar, R. Greenstadt, and C. Diaz, "How unique is your. onion? an analysis of the fingerprintability of tor onion services," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[54] M. V. Pedersen and A. Askarov, "From trash to treasure: timing-sensitive garbage collection," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017.

[55] M. V. Pedersen and A. Askarov, "Static enforcement of security in runtime systems," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019.

[56] M. Perry, E. Clark, S. Murdoch, and G. Koppen, "The design and implementation of the tor browser [draft]," https://2019.www.torproject.org/projects/torbrowser/design/, June 2018, (Accessed on 08/03/2023).

[57] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "SafeBricks: Shielding network functions in the cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/poddar

[58] D. E. Porter, M. D. Bond, I. Roy, K. S. McKinley, and E. Witchel, "Practical fine-grained information flow control using laminar," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 37, no. 1, 2014.

[59] E. Prouff and M. Rivain, "Masking against Side-Channel Attacks: A Formal Security Proof," in *Advances in Cryptology – EUROCRYPT 2013*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 7881.

[60] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-51. Fukuoka, Japan: IEEE Press, Oct. 2018.

[61] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. Phoenix, Arizona: AMC, Jun. 2019.

[62] W. Rafnsson, L. Jia, and L. Bauer, "Timing-sensitive noninterference through composition," in *International Conference on Principles of Security and Trust*. Springer, 2017.

[63] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015.

[64] A. Rane, C. Lin, and M. Tiwari, "Secure, precise, and fast floating-point operations on x86 processors," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016.

[65] P. Rohatgi, "Improved Techniques for Side-Channel Analysis," in *Cryptographic Engineering*. Springer, Boston, MA, 2009.

[66] D. Schultz and B. Liskov, "Ifdb: decentralized information flow control for databases," in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[67] M. Schwarzl, P. Borrello, A. Kogler, K. Varda, T. Schuster, M. Schwarz, and D. Gruss, "Robust and scalable process isolation against spectre in the cloud," in *European Symposium on Research in Computer Security*. Springer, 2022.

[68] A. Shusterman, A. Agarwal, S. O'Connell, D. Genkin, Y. Oren, and Y. Yarom, "{Prime+Probe} 1, {JavaScript} 0: Overcoming Browser-based {Side-Channel} Defenses," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[69] A. Shusterman, Z. Avraham, E. Croitoru, Y. Haskal, L. Kang, D. Levi, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Website Fingerprinting Through the Cache Occupancy Channel and its Real World Practicality," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, Sep. 2021.

[70] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust Website Fingerprinting Through the Cache Occupancy Channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019.

[71] R. Sinha, S. Gaddam, and R. Kumaresan, "LucidiTEE: A tee-blockchain system for policy-compliant multiparty computation with fairness," Cryptology ePrint Archive, Paper 2019/178, 2019, https://eprint.iacr.org/2019/178. [Online]. Available: https://eprint.iacr.org/2019/178

[72] D. Stefan, P. Buiras, E. Z. Yang, A. Levy, D. Terei, A. Russo, and D. Mazières, "Eliminating cache-based timing attacks with instruction-based scheduling," in *European Symposium on Research in Computer Security*. Springer, 2013.

[73] D. Stefan, A. Russo, P. Buiras, A. Levy, J. C. Mitchell, and D. Mazieres, "Addressing covert termination and timing channels in concurrent information flow systems," *ACM SIGPLAN Notices*, vol. 47, no. 9, 2012.

[74] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An Extremely Simple Oblivious RAM Protocol," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013.

[75] J. Szefer, "Survey of microarchitectural side and covert channels, attacks, and defenses," *Journal of Hardware and Systems Security*, vol. 3, no. 3, 2019.

[76] J. Thibault and A. Askarov, "Improving language-based predictive mitigation for information-flow security," *arXiv preprint arXiv*.

[77] M. Tiwari, X. Li, H. M. G. Wassel, F. T. Chong, and T. Sherwood, "Execution leases: A hardware-supported mechanism for enforcing strong non-interference," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: Association for Computing Machinery, 2009.

[78] M. Vassena, J. Breitner, and A. Russo, "Securing concurrent lazy programs against information leakage," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 2017.

[79] M. Vassena, A. Russo, P. Buiras, and L. Waye, "Mac a verified static information-flow control library," *Journal of logical and algebraic methods in programming*, vol. 95, 2018.

[80] M. Vassena, G. Soeller, P. Amidon, M. Chan, J. Renner, and D. Stefan, "Foundations for parallel information flow control runtime systems." in *POST*, 2019.

[81] T. Wang, "High precision open-world website fingerprinting," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[82] X. S. Wang, K. Nayak, C. Liu, T. H. Chan, E. Shi, E. Stefanov, and Y. Huang, "Oblivious data structures," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[83] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting cache attacks via cache set randomization," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019.

[84] M. Wu, S. Guo, P. Schaumont, and C. Wang, "Eliminating timing side-channel leaks using program repair," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3213846.3213851

[85] W. Wu and B. Ford, "Deterministically deterring timing attacks in deterland," *arXiv preprint arXiv:1504.07070*, 2015.

[86] Q. Xiao, M. K. Reiter, and Y. Zhang, "Mitigating storage side channels using statistical privacy mechanisms," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2810103.2813645

[87] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*, 2015.

[88] D. Zagieboylo, G. E. Suh, and A. C. Myers, "Using information flow to design an isa that controls timing channels," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019.

[89] D. Zhang, A. Askarov, and A. C. Myers, "Predictive mitigation of timing channels in interactive systems," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2046707.2046772

[90] D. Zhang, A. Askarov, and A. C. Myers, "Language-based control and mitigation of timing channels," in *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*, 2012.

[91] T. Zhang, F. Liu, S. Chen, and R. B. Lee, "Side channel vulnerability metrics: The promise and the pitfalls," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. Tel-Aviv, Israel: AMC, Jun. 2013.

[92] X. Zhang, J. Hamm, M. K. Reiter, and Y. Zhang, "Statistical privacy for streaming traffic." in *NDSS*, 2019.

[93] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM Side Channels and Their Use to Extract Private Keys," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012.

[94] M. Zhao, M. Gao, and C. Kozyrakis, "ShEF: Shielded enclaves for cloud fpgas," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2022. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3503222.3507733

[95] H. Zheng and O. Arden, "Secure distributed applications the decent way," in *Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems*, ser. ASSS '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3457340.3458304

[96] Y. Zhou, S. Wagh, P. Mittal, and D. Wentzlaff, "Camouflage: Memory Traffic Shaping to Mitigate Timing Attacks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2017.