Brutus: Refuting the Security Claims of the Cache Timing Randomization Countermeasure Proposed in CEASER

Rahul Bodduna, Vinod Ganesan, Patanjali SLPSK, Kamakoti Veezhinathan, and Chester Rebeiro

Abstract—Cache timing attacks are a serious threat to the security of computing systems. It permits sensitive information, such as cryptographic keys, to leak across virtual machines and even to remote servers. Encrypted Address Cache, proposed by CEASER – a best paper candidate at MICRO 2018 – is a promising countermeasure that stymies the timing channel by employing cryptography to randomize the cache address space. The author claims strong security guarantees by providing randomization both spatially (randomizing every address) and temporally (changing the encryption key periodically). In this letter, we point out a serious flaw in their encryption approach that undermines the proposed security guarantees. Specifically, we show that the proposed Low-Latency Block Cipher, used for encryption in CEASER, is composed of only linear functions which neutralizes the spatial and temporal randomization. Thus, we show that the complexity of a cache timing attack remains unaltered even with the presence of CEASER. Further, we compare the encryption overheads if CEASER is implemented with a stronger encryption algorithm.

1 Introduction

Cache timing attacks are a potent side-channel technique exploiting the shared cache memories present in modern day microprocessors. In a typical attack, the attacker executes a spy program in the same CPU as a victim application. The spy is designed to perform a set of memory operations and monitor the execution time. The execution time, which varies due to interference with the victim in the shared cache memory, can be used to reveal sensitive information about the victim. These attacks have been used in a variety of applications, such as, breaking cryptographic ciphers [3], [13], undermining Operating System security [10], fingerprinting websites [17], and logging keystrokes [16]. They have been applied on cloud computing platforms [16] and to attack remote computers [12]. These attacks rely on the fixed mapping from memory addresses to cache sets. Thus, a conflict between the spy and victim in the cache, can partially reveal information about the victim's memory operations, which in turn is correlated to sensitive information.

A popular approach to counter cache timing attacks is based on randomizing cache addresses thus breaking the fixed mapping in the cache memory. CEASER [15] proposed at MICRO 2018, is one such randomization countermeasure which provides randomization both spatially and temporally. Every address in the cache is randomized using an encryption with a secret key that is generated internally in the hardware. The encryption causes memory addresses that are spatially correlated to get mapped to random cache sets. This makes it difficult for an attacker to identify addresses that conflict with the victim process. Temporal randomization is achieved by changing the encryption key periodically. Thus, the same memory address would get mapped to different cache sets over time. Temporal randomization would ensure that, even if the attacker manages to identify a set of addresses that conflict with an address in the victim's memory space, the change in encryption key, would shuffle the cache set mapping, thus invalidating the conflict. CEASER also provides a security evaluation, where they claim that the complexity of finding such a conflicting set of addresses, called an *eviction set*, is $O(L^2)$, where L is the number of sets present in the cache. In ISCA 2019, an extension of CEASER called CEASER-S [14] was proposed to strengthen the countermeasure against newly found attacks [18], [19].

Central to the security of CEASER and CEASER-S is the encryption algorithm. A flaw in the encryption algorithm can undermine the entire scheme permitting the spy process to identify eviction sets and there by discover the victim's memory access patterns. Further, since the encryption algorithm is in the critical path in the cache memory, it has to have low overheads. To satisfy these requirements, CEASER proposes a Low-Latency Block Cipher (LLBC) based on a 4-round Feistel Network [8] and an 80-bit key. CEASER-S also reuses the same LLBC algorithm to randomize the cache addresses. In order to achieve the crucial requirement of low latency, the authors of CEASER have **(1)** reduced the number of rounds of the cipher to 4 (compared to traditional ciphers which have 16 to 18 rounds [2], [7]); and, **(2)** simplified the Substitution and Permutation function in the cipher to use only linear functions. Traditional ciphers use a mixture of linear and non-linear functions to increase security.

Due to this linear nature of the LLBC algorithm, the encrypted output can be represented by linear functions of the input and key. In this paper we present a major flaw in the design of the Low-Latency Block Cipher that can nullify all the security guarantees provided by CEASER and CEASER-S. We show that the linearity of CEASER's cipher breaks the temporal and spatial randomization that CEASER provides. If an attacker finds an eviction set, the collision persists even after the cipher's key is changed. In summary, our key contributions are as follows.

- We show that CEASER fails to protect against cache timing attacks due to the linear properties of the LLBC cipher used.
- We further show that, due to the linearity in the cipher algorithm, the complexity of finding an eviction set is the same as when there is no randomization present.
- While randomization of cache addresses, if done properly, is a promising countermeasure for cache timing attacks, we are still far from finding a low-overhead solution. We show that replacing the LLBC with a standard lightweight cipher, such as PRINCE [4] can lead to considerable overheads on both FPGAs and ASICs.

This paper is structured as follows: The background is presented in Section 2. A detailed illustration of the weakness in CEASER is discussed in Section 3 followed by experimental validation of the PRINCE cipher. The final section concludes the paper.

2 Background

This section provides the necessary background on cache timing attacks, block ciphers and the LLBC algorithm used in CEASER and CEASER-S.

Cache Timing Attacks. In cache timing attacks that target the Last Level Cache (LLC), a crucial aspect for the attacker is to find the minimum set of memory addresses in the spy process that can evict a targeted victim address from the LLC. Finding this set, called the *eviction Set*, is difficult due to two reasons. First, the LLC is physically addressed and physical addresses are typically hidden from user space programs. Second, most LLCs are designed with slices and have an undisclosed hash function that distributes addresses across the slices [11].

- *The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, Tamil Nadu 600036, India. E-mail: {rahulb, vinodg, slpskp, kama, chester}@cse.iitm.ac.in.*

Manuscript received 9 Oct. 2019; revised 22 Nov. 2019; accepted 11 Dec. 2019. Date of publication 6 Jan. 2020; date of current version 27 Jan. 2020.
(Corresponding author: Rahul Bodduna.)
Digital Object Identifier no. 10.1109/LCA.2020.2964212

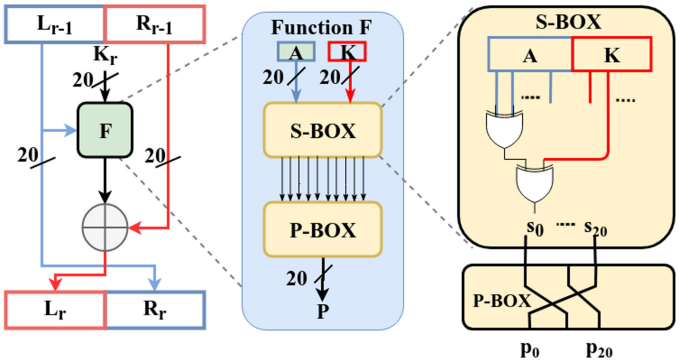

Fig. 1. An Overview of the Low-Latency Block Cipher (LLBC) used in CEASER [15] and CEASER-S [14].

Given a targeted victim address A_v, an eviction set is identified in two steps. First, a memory access to A_v is done and the data is stored in the LLC. Then, randomly chosen memory addresses from the spy's memory are accessed one after the other until A_v is evicted from the cache. This set of addresses, in the worst case, can be as large as the entire cache. In the second step, the set is trimmed to comprise of the smallest number of addresses that can evict A_v. This minimal set is called the eviction set.

If the total number of cache lines in a W-way set associative LLC is L, finding the eviction set require $O(L^2)$ memory accesses, while the size of the eviction set is $O(W)$.

Block Ciphers. A block cipher is an encryption algorithm that maps a block of plaintext to a ciphertext. A typical block cipher operates over multiple rounds, with each round comprising confusion and diffusion functions. Diffusion is done by linear functions which help distribute the plaintext from one byte over several bytes. Linear functions are represented as follows:

$$\mathcal{A}\Big(x_1, x_2, \ldots, x_n\Big) = \alpha_1 \cdot x_1 \oplus \alpha_2 \cdot x_2 \ldots \alpha_n \oplus x_n. \tag{1}$$

Confusion operations reduce the correlation between the input and the output and are often implemented by non-linear substitution boxes (S-Boxes). A non-linear function is represented as follows:

$$\mathcal{A}\Big(x_1, x_2, \ldots, x_n\Big) = \oplus_{j=1}^{n} \alpha_j \prod_{z \in Z} x_z, \tag{2}$$

where x_1, x_2, …, x_n are function inputs, α_1, α_2, …, α_n are constant binary coefficients and $Z \subseteq \{1, 2 \ldots, n\}$.

Low-Latency Block Cipher used in CEASER. CEASER and CEASER-S employ a Low-Latency Block Cipher using a 4-round Feistel structure to randomize addresses in the cache memory. The 40-bit memory address entering the cache is split into two equal parts $L^{(0)}$ and $R^{(0)}$. In each round r ($1 \leq r \leq 4$), a 20-bit round key ($K^{(r)}$) is used and the following operations are performed

$$\begin{aligned} L^{(r)} &= R^{(r-1)} \oplus \mathcal{F}(L^{(r-1)}, K^{(r)}), \\ R^{(r)} &= L^{(r-1)}, \end{aligned} \tag{3}$$

where function $\mathcal{F}$ comprises a substitution followed by a permutation operation as shown in Fig. 1. The substitution operation takes $K^{(r)}$ and $L^{(r-1)}$ as inputs and outputs a 20-bit result. The output of the substitution operation is then shuffled using a permutation operation $P^{(r)}$. The substitution and permutation operations are linear, thus, the $\mathcal{F}$ function in the LLBC cipher is linear. Half the bits of a round's output is derived from this $\mathcal{F}$ function while the other half is obtained from the round's input. Thus, every bit of the round's output is a linear function of the secret key and the round input. The output at end of 4 rounds ($r = 4$) is considered the encrypted address and used as an index to the cache sets.

3 The Pitfall in CEASER's LLBC Design

In this section we show that the complexity of mounting an attack on CEASER is same as CEASER without remapping. We first show that the temporal and spatial randomization with respect to each LLBC that CEASER provides is broken due to the linear nature of the operations used in the LLBC algorithm. As a result, even though CEASER switches between the two LLBCs to randomize cache mappings, attacks can still be easily mounted.

Linearity of the LLBC Algorithm. The output of the first round is linearly related to the address sent to the cache. Each output bit of a subsequent round is a linear function of the previous round's output. Thus, the output of the second, third, and fourth rounds are linearly related to the address sent to the cache. Each bit (o_i, $1 \leq i \leq 40$) of the encrypted address can thus be represented by linear equations of the following form:

$$\begin{aligned} o_i = \mathcal{C}(A) \oplus \mathcal{D}(K) = c_1 a_1 \oplus c_2 a_2 \oplus \cdots \oplus c_{40} a_{40} \oplus \\ d_1 k_1 \oplus d_2 k_2 \oplus \cdots \oplus d_{80} k_{80}, \end{aligned} \tag{4}$$

where A is the address sent to the cache; K an 80-bit string of concatenated round keys ($K = K^{(1)}||K^{(2)}||K^{(3)}||K^{(4)}$); a_j and k_l denote the jth ($1 \leq j \leq 40$) and lth ($1 \leq l \leq 80$) bit of A and K respectively. The functions $\mathcal{C}$ and $\mathcal{D}$ are linear with coefficients c_j and d_l respectively that are chosen randomly from the set {0, 1} during design.

Refuting CEASER's Claim on Temporal Randomization. The objective of CEASER is to randomize cache addresses using the LLBC, making it difficult for a spy process to find a set of memory addresses that collide with a victim's memory access. To further strengthen the scheme, LLBC's secret key is changed periodically. Thus, even if the spy manages to find addresses that collide with the victim's address, the change in key would change the randomization, thus invalidating the collision. The benefits of finding a collision is therefore short lived.

In this section, we show that changing the LLBC key, in fact, does not invalidate the collision. The colliding spy addresses continue to collide with the victim address even after the key is changed. To see this, assume that at some instant, two addresses A_s (in the spy address space) and A_v (in the victim address space) collide in the cache. A collision means that the corresponding encrypted addresses map to the same cache set. Consider a single bit o_i ($1 \leq i \leq indexBits$) of the encrypted addresses, where *indexBits* is the number of address bits used to index into the cache. From Equation (4), we obtain the following equation

$$\mathcal{C}(A_s) \oplus \mathcal{D}(K) = o_i = \mathcal{C}(A_v) \oplus \mathcal{D}(K). \tag{5}$$

Thus,

$$\mathcal{C}(A_s) = \mathcal{C}(A_v). \tag{6}$$

We see that the key has no influence on the collision. Thus the collision persists even after the key is changed.

We illustrate the *key invariance* and its impact with an example in Fig. 2. Assume a Feistel Network whose output bits (O_0, O_1, O_2, O_3) are described in the figure. The bits O_1 and O_0 are used to index into cache set. Assume two 4-bit addresses $A_v = (0000)_2$ and $A_s = (1110)_2$ and at some instant of time, the key K_1 used by the Feistel Network is $(1101)_2$. On encryption, A_v and A_s result in $(0011)_2$ and $(0111)_2$ respectively. The lower two bits of both encrypted addresses is $(11)_2$. Therefore, the two addresses map to the same cache set. When the key is changed to $K_2 = (0011)_2$, the addresses A_v and A_s now get encrypted to $(1000)_2$ and $(1100)_2$ respectively. While we notice that the cache set change from $(11)_2$ to $(00)_2$, the collision between A_v and A_s is maintained, *i.e* both A_v and A_s continue to map to the same cache set. Thus, if an attacker can find a set of addresses that collide in the cache with a victim address,

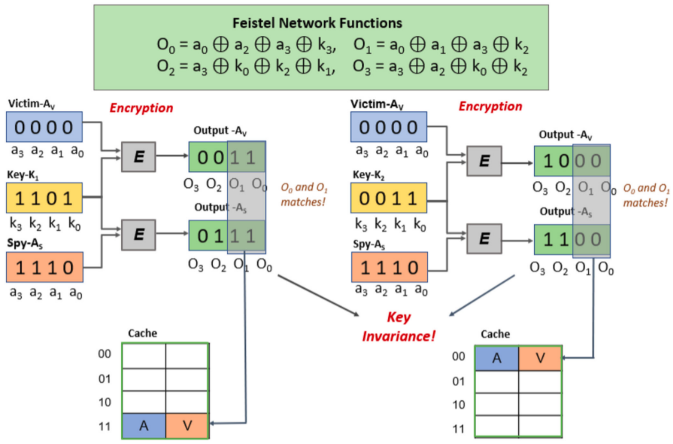

Fig. 2. Illustration of a key dependent Feistel Network showing key invariance.

the collision continues to persist with the change in encryption key giving ample time for the attacker to mount a cache timing attack.

Refuting CEASER's Claim on Spatial Randomization. Due to the cache address encryption and the periodic change in key, CEASER claims that the probability of finding an eviction set is very low – in the order of 100 years. In this section, we show that, it takes a one-time effort of $O(L^2)$ accesses, where L is the number of cache lines in the LLC, to partially reverse engineer the $\mathcal{C}(A)$ used in Equation (4). As discussed in CEASER, finding an eviction set is sufficient to break spatial randomization.

An essential requirement to build an eviction set is to identify addresses that map to the same cache set. In Equation (4), the encrypted output bit o_i depends on a subset of the address bits a_1 to a_{40}. This subset is defined by the binary coefficients c_1 to c_{40} respectively. Flipping an even number of address bits in the subset will not alter the encrypted output bit o_i. Flipping even number of address bits in every encrypted output bit o_i $(1 \leq i \leq indexBits)$ would form a new address which gets mapped to the same cache set as before, thus aiding in creation of the eviction set. Fig. 3 provides an example of this phenomenon. Assume a physical address $A = (0000)_2$ that is encrypted with a key $K = (1101)_2$ to produce the encrypted address $(1001)_2$. Flipping a_0, a_1 and a_2 produces a different address $(1110)_2$ but the encrypted address $(0001)_2$ maps to the same cache set as $(1001)_2$. Thus, $(0000)_2$ ad $(1110)_2$ can be part of the same eviction set.

The difficulty in this scheme is to identify the bits to be flipped so that the encrypted address continues to be mapped to the same cache set. We call these bits the *Invariant Bits*. Finding the Invariant Bits is system specific as Equation (4) is defined at the design time and kept secret. We can follow a similar approach as the eviction set algorithm described in [11] (and described in Section 2) to find the Invariant Bits. Once found, given any physical address, an eviction set can be created in $O(1)$ by flipping the Invariant bits. Due to the key invariance property (Fig. 2), finding the Invariant Bits needs to be done only once for a system.

Attacking CEASER when Two Different LLBCs are Used. To achieve gradual remapping CEASER suggests the use two parallel

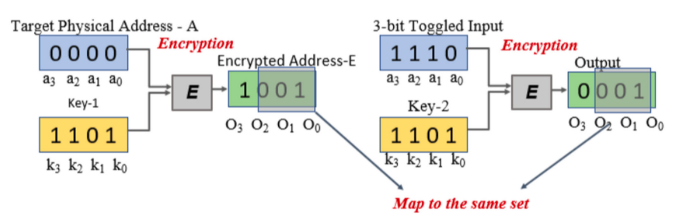

Fig. 3. Illustration of a invariance on toggling input bits. The Feistel function is same as used in Fig. 2

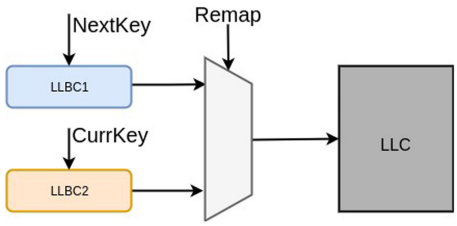

Fig. 4. Organization of LLBC circuits in CEASER.

LLBC circuits as shown in Fig. 4 to reduce overheads of remapping. Although not explicitly mentioned in [15], the two LLBC circuits may have different encryption algorithms. In this section we show that CEASER breaks even with such a configuration.

Due to the presence of the two different LLBC algorithms, there would be two eviction sets corresponding to an address A_v; one eviction set for each LLBC. At any given instant, the attacker would not know which LLBC is used to encrypt A_v, thus needs to try both eviction sets. Finding the pair of eviction sets can be done using techniques similar to those described in [11] with additional checks to ensure that each eviction set corresponds to a different LLBC.

Fig. 5 shows the time required to construct the eviction set with different number of cache sets and associativity and CEASER configured with two LLBC circuits as shown in Fig. 4. We assume that the physical offset is 30-bits. As the associativity of the cache increases, the size of the eviction set also increases and therefore takes longer to construct. Further, it takes longer to locate the address A_v, thus a linear increase in time with cache size. We experimented by exhaustively toggling a set of 5-bits of the address for each cache configuration and estimated the time taken to find an eviction set. We see that time taken to find the eviction set is well with in the practical limits.

4 Replacing the LLBC for Better Security

While randomization is an excellent proposal to mitigate several cache timing attack variants, the limitation of CEASER is the use of an encryption algorithm without non-linear operations. We evaluate a promising LLBC candidate called PRINCE [4] and compare it with CEASER's LLBC. PRINCE is a 12 round block cipher with an FX construction [9] and non-linear S-Box operations.

We implemented CEASER's LLBC and compared it with PRINCE. Table 1 shows the results for the two implementations along with a 7-round PRINCE implementation. All the designs were synthesized on Xilinx Artix-7 for FPGAs and at Intel's 22 nm technology node for ASICs using Synopsis Design Compiler. We note considerable overheads, of upto $7.5\times$ compared to CEASER's LLBC on ASIC. The area requirements increase by over $22\times$ on the same platform. We also experimentally verified that PRINCE does not exhibit the spatial and temporal vulnerabilities that CEASER suffers. To estimate run time overheads with PRINCE we use Sniper [5] (gainestown configuration), and we run memory intensive multi-threaded Splash2 workloads (Misses Per Kilo

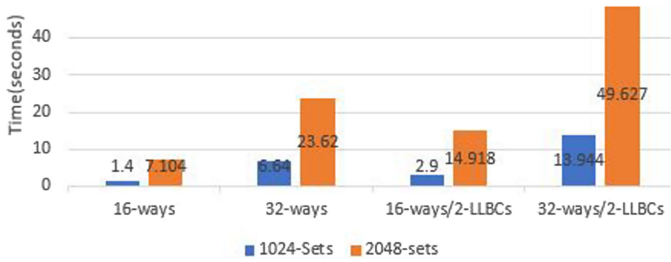

Fig. 5. Time to find eviction sets for varying cache configuration.

TABLE 1
Overheads of CEASER's LLBC with a Stronger Cipher PRINCE on ASIC and FPGAs

	CEASER's LLBC	PRINCE (12 rounds)	PRINCE (7 rounds)
Delay(FPGA)	3.1 ns	14.7 ns	8.7 ns
Area(FPGA LUTs)	135	863	569
Latency(FPGA)	1×	4.7×	2.8×
Delay(ASIC)	250 ps	1890 ps	980 ps
Area(ASIC Cells)	237	5327	3737
Latency(ASIC)	1×	7.5×	3.92×

We define latency as the ratio of the delay incurred due to using any encryption to the delay incurred due to using the Feistel Network proposed in CEASER.

Instructions (MPKI) $>$ 10). We found these memory intensive workloads showed a performance degradation of 3.2 percent on an average. We also observe that higher the MPKI, larger the performance impact. Hence performance impact will be more pronounced with respect to CloudSuite [6] and TPC-C [1] applications which have higher MPKI. We intend to perform this rigorous analysis of evaluating the fundamental tradeoff between strength of the cipher and processor's performance with Encrypted Address Caches as a future work.

5 Conclusion

While the cache address randomization proposed in CEASER [15] is a promising solution to counter cache timing attacks, it is still an open challenge to design efficient encryption algorithms that can perform the address randomization securely. The current algorithm proposed in [15] has a serious flaw due to the absence of non-linear components. This flaw entirely compromises the security provided by cache address encryption. A direct replacement of CEASER's cipher with standard low-latency block ciphers like PRINCE, that does not suffer from this flaw, has significant performance penalties. Such performance overheads, of up to 7.5×, is unacceptable in high-speed microprocessors. This opens up a need for specialized low-latency encryption techniques that are exclusively designed for cache address encryption, that can provide the security guarantees with acceptable performance overheads.

References

[1] "TPC-C," Accessed: 2015. [Online]. Available: http://www.tpc.org/tpcc/default.asp
[2] K. Aoki *et al.*, "Camellia: A 128-bit block cipher suitable for multiple platforms design andanalysis," in *Proc. Int. Workshop Sel. Areas Cryptography*, 2000, pp. 39–56.
[3] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2006, pp. 201–215.
[4] J. Borghoff *et al.*, "PRINCE – a low-latency block cipher for pervasive computing applications," in *Proc. Theory Appl. Cryptology Inf. Security*, 2012, pp. 208–225.
[5] T. E. Carlson *et al.*, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2011, pp. 1–12.
[6] M. Ferdman *et al.*, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proc. 17th Int. Conf. Architectural Support Program. Languages Operating Syst.*, 2012, pp. 37–48.
[7] FIPS46–3, "Data encryption standard," *Federal Inf. Process. Standards Publ.*, pp. 46–3, 1999.
[8] J. Katz *et al.*, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC.
[9] J. Kilian *et al.*, "How to protect DES against exhaustive key search," in *Proc. Annu. Int. Cryptology Conf.*, 1996, pp. 252–267.
[10] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 973–990.
[11] F. Liu *et al.*, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 605–622.
[12] Y. Oren *et al.*, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 1406–1418.
[13] C. Percival, "Cache missing for fun and profit (2005)," 2015. [Online]. Available: http://www.daemonology.net/papers/htt.pdf
[14] M. Qureshi, "New attacks and defense for encrypted-address cache," in *Proc. 46th Int. Symp. Comput. Architecture*, 2019, pp. 360–371.
[15] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2018, pp. 775–787.
[16] T. Ristenpart *et al.*, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. 16th ACM Conf. Comput. Commun. Security*, 2009, pp. 199–212.
[17] A. Shusterman *et al.*, "Robust website fingerprinting through the cache occupancy channel," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 639–656.
[18] W. Song and P. Liu, "Dynamically finding minimal eviction sets can be quicker than you think for side-channel attacks against the LLC," in *Proc. 22nd Int. Symp. Res. Attacks Intrusions Defenses*, 2019, pp. 427–442.
[19] P. Vila *et al.*, "Theory and practice of finding eviction sets," *IEEE Symp. Secur. Privacy*, pp. 39–54, 2018.