Scientia Iranica D (2022) 29(4), 1949–1956

SCIENTIA IRANICA

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
http://scientiairanica.sharif.edu

Modified cache template attack on AES

M. Esfahani[a], H. Soleimany[b], and M.R. Aref [c,*]

a. *Department of Mathematics, Karaj Branch, Islamic Azad University, Karaj, Iran.*
b. *Cyberspace Research Institute, Shahid Beheshti University, Tehran, Iran.*
c. *Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran.*

Received 12 October 2019; received in revised form 11 October 2020; accepted 7 December 2020

KEYWORDS
Cache memory;
Flush+Reload attack;
Inclusive memory;
AES;
Cache template attacks;
Automation.

Abstract. CPU caches are powerful sources of information leakage. To develop practical cache-based attacks, the need for automation of the process of finding exploitable cache-based side-channels in computer systems is felt more than ever. Cache template attack is a generic technique that utilizes Flush+Reload attack in order to automatically exploit cache vulnerability of Intel platforms. Cache template attack on the T-table-based AES implementation consists of two phases including the profiling phase and key exploitation phase. Profiling is a preprocessing phase to monitor dependencies between the secret key and behavior of the cache memory. In addition, the addresses of T-tables can be obtained automatically. At the key exploitation phase, Most Significant Bits (MSBs) of the secret key bytes are retrieved by monitoring the exploitable addresses. This study proposed a simple yet effective searching technique, which accelerates the profiling phase by a factor of utmost 64. In order to verify the theoretical model of our technique, the mentioned attack on AES was implemented. The experimental results revealed that the profiling phase runtime of the cache template attack was approximately 10 minutes, while the proposed method could speed up the running of this phase up to almost 9 seconds.

1. Introduction

Cryptographic algorithms and protocols alone fail to provide data security; therefore, they require a digital platform to run securely and efficiently. Security evaluation of the cryptographic algorithms against side-channel attacks is one of the most important challenges in the field of applied cryptography. Unlike the mathematical analyses that consider the structural weaknesses in the cryptographic primitives, side-channel attacks use data leaking from the implementation of the cryptographic algorithms.

Timing variation during the run-time program is one of the most important sources of information leakage in the timing channels. Memory access and presence of branch in the programs are costly at the runtime. For this reason, modern processors use cache memories and branch predictors to reduce this cost. Such an optimization at the runtime would lead to timing variations. Easy measurement without the need for specific hardware tools for this purpose is among the specific features of timing side-channel attacks. The cache-based side channel attacks distinguish between cache hit and cache miss events by measuring the execution time of the target cryptographic algorithm. The execution time difference between the cache hit and cache miss leads to information leakage. Cache-based side-channel attacks are classified into three categories namely the time-driven, trace-driven, and access-driven attacks. In the time-driven attacks [1,2], the attacker does not have access to the cache and it knows the capacity of the cache memory lines. In addition, the attacker should retrieve the secret

*. *Corresponding author. Tel./Fax: +98 21 66165908*
E-mail address: aref@sharif.ac.ir (M.R. Aref)

doi: 10.24200/sci.2020.54534.3797

key by only measuring the cryptosystem runtime. In the access-driven attacks, the attacker is able to evict or reload data from the cache memory [3–5]. Access-driven attacks are classified into synchronous and asynchronous categories. While the attackers in synchronous attacks are able to trigger encryption or decryption, they act as non-privileged adversaries in parallel to the victim in asynchronous attacks [6–8]. In trace-driven attacks, the attacker observed a series of cache misses and cache hits during encryption [9].

The first covert channel based on the cache memory was proposed by Hu [10]. Kelsey believed that types of attacks done based on cache hit ratio in the ciphers with large S-boxes were likely to happen [11]. Later, Tsunoo took into account the cache-based side-channel attacks on the implementation of the ciphers with large lookups and obtained the first results from the experimental attacks on the block ciphers such as DES [12]. Bernstein used the aggregate number of cache hits and misses through indirect measurements of the total execution time of the encryption process in order to attack AES for the first time [13]. Since then, several practical time-driven cache attacks on AES have been proposed [14,15]. Percival et al. were the pioneers in the access-driven attacks on RSA and AES [16]. Yarom and Falkner [17] proposed the Flush+Reload attack and successfully applied the attack on the implementation of RSA. Ronen et al. [18,19] performed the Flush+Reload attack on the targets using the last level cache in the virtualized environments. Then, they employed the Flush+Reload technique to retrieve all 16 bytes of AES in the native and cross-VM environments, respectively [20,21]. In the next year, Gülmezoğlu et al. [22] improved the attack [21] by predicting the possible candidates for the last round of AES, thus reducing the attack noise.

In a majority of the proposed attacks, the attacker should identify the vulnerabilities manually, which is a considerable limitation. In response to this challenge, Gruss et al. [23] proposed cache template attacks. The attack makes use of the Flush+Reload technique in order to automatically exploit the cache-based vulnerabilities in a program running on architecture with shared inclusive last-level caches.

1.1. Our contribution

The cache-template attack on the T-table based implementation of the AES proposed in [23] performs both profiling and exploitation phases automatically. The high runtime of the profiling phase is an important limiting factor in the proposed attack. Measuring the cache-hit ratio is the most expensive step in the attack case.

In order to increase the runtime speed at the profiling phase, the current research proposed a simple yet efficient method to measure the cache-hit ratio for each address of the attacked binary and construct the profile.

1.2. Outline

This paper is organized as follows. Section 2 presents the background information. Section 3 defines the cache template attacks. Section 4 presents an overview of the proposed technique as well as the experimental results. Finally, Section 5 concludes the study.

2. Background

2.1. CPU Caches

Cache is an essential feature of modern architecture that increases the speed of memory access by keeping the recently accessed instructions and data. The cache memory is organized as multiple cache sets, each consisting of a fixed number of cache lines [24]. Each cache line is split into a *tag, index*, and *block offset*. The index is used to map the specific memory locations in the cache memory sets. The most significant bits of the address determine the tag, which is used to uniquely identify a specific cache line in a cache set. The block offset identifies a particular location within a cache line.

In order to bridge the gap between the data retrieval and processor speeds, modern processors exploit a hierarchy in the cache structure. Closest to the core is the L1 cache which consists of separate parts for data and instructions while other levels are unified. Down to the Last-Level Cache (LLC), the cache level gets larger and slower. The last-level cache is generally shared between the cores. In most of Intel processors, the cache memory has three levels and LLC is inclusive, which means all data in the L1 and L2 caches are also present within the L3 cache [25]. L3 cache is shared among all cores, and the inclusive cache is used to apply the Flush+Reload attack [21,22,26,27], which will be described in the next section.

2.2. Flush+Reload attack

In Flush+Reload attack on the Intel system, an attacker flushes the cache memory using the CLFLUSH instructions. The Flush+Reload attack on the cryptographic algorithms makes use of the shared memory and library features in the L3 cache between the attacker and victim program. The functions of the Flush+Reload are illustrated in the following:

1. The attacker maps the shared library (or binary) into the virtual address space and accesses it to facilitate loading into the cache;
2. The attacker flushes the shared library from the cache and waits an appropriate amount of time for the victim to use (or not use) the memory locations that he has already flushed;
3. Once the victim is scheduled, the attacker reloads

the previously flushed shared library and measures the load time.

It was mainly observed that in case the victim could not access the data flushed in the second step, the data would not be available in the cache memory and consequently, the attacker would measure high latency. There are two main reasons why the Flush+Reload method is more powerful than the previous access-driven attacks. First, unlike previous attacks initiated based on the cache set, the attacker here has access to the cache line in the Flush+Reload attack, which leads to an increase in the accuracy. Second, the Flush+Reload attack is a cross-core attack as the L3 cache is shared among all processor cores. For this reason, Flush+Reload method has been used in many cache-based attacks in recent years [28–32].

2.3. Memory access in AES implementations

Advanced Encryption Standard (AES) has been adopted by the U.S. government as an encryption standard [33]. It is characterized by a Substitution-Permutation Network (SPN) structure with the fixed block size of 128 bits and key size of 128, 192 or 256 bits. The current study took into account an attack on the AES-128. AES operates on a 4×4 order array of bytes called the state matrix, and most calculations are done in $GF(2^8)$. AES-128 has ten rounds, each congaing four types of transformation namely SubByte, ShiftRows, MixColumns, and AddRoundKey. Exceptionally, the last round does not have MixColumns.

Different methods have been used in both hardware and software to increase the speed and efficiency of software implementation. Given that the SubBytes is the most expensive type to implement, the lookup table in the software implementation is ideal to run this operation. However, a well-known method T-table implementation [33] has been adopted based on several crypto libraries such as OpenSSL which precomputes the round function. In the T-table implementation, the four look-up tables are as follows:

$$T_0(z) = \begin{bmatrix} 02.S(z) \\ S(z) \\ S(z) \\ 03.S(z) \end{bmatrix}, \qquad T_1(z) = \begin{bmatrix} 03.S(z) \\ 02.S(z) \\ S(z) \\ S(z) \end{bmatrix},$$

$$T_2(z) = \begin{bmatrix} S(z) \\ 03.S(z) \\ 02.S(z) \\ S(z) \end{bmatrix}, \qquad T_3(z) = \begin{bmatrix} S(z) \\ S(z) \\ 03.S(z) \\ 03.S(z) \end{bmatrix}. \tag{1}$$

Each table maps a byte z to a 32-bit value. Consequently, the size of each T-table is 1024 bytes. If we assume that the size of each cache line is 64 bytes, 16-cache lines are required to store one T-table. Based on the T-tables presented in Eq. (1), we can express the first nine rounds of AES, as described in Eq. (2).

$$\begin{aligned} &T_0[s_0^{(r)}] \oplus T_1[s_5^{(r)}] \oplus T_2[s_{10}^{(r)}] \oplus T_3[s_{15}^{(r)}] \\ &\qquad \oplus [k_0^{(r)} k_1^{(r)} k_2^{(r)} k_3^{(r)}]||, \\ &T_0[s_4^{(r)}] \oplus T_1[s_9^{(r)}] \oplus T_2[s_{14}^{(r)}] \oplus T_3[s_3^{(r)}] \\ &\qquad \oplus [k_4^{(r)} k_5^{(r)} k_6^{(r)} k_7^{(r)}]||, \\ &T_0[s_8^{(r)}] \oplus T_1[s_{13}^{(r)}] \oplus T_2[s_2^{(r)}] \oplus T_3[s_7^{(r)}] \\ &\qquad \oplus [k_8^{(r)} k_9^{(r)} k_{10}^{(r)} k_{11}^{(r)}]||, \\ &T_0[s_{12}^{(r)}] \oplus T_1[s_1^{(r)}] \oplus T_2[s_6^{(r)}] \oplus T_3[s_{11}^{(r)}] \\ &\qquad \oplus [k_{12}^{(r)} k_{13}^{(r)} k_{14}^{(r)} k_{15}^{(r)}], \end{aligned} \tag{2}$$

where $s_i^{(r)}$ represents the ith byte of the state in the rth round in which $0 \le r \le 9$, and $0 \le i \le 15$. The final round cannot use the tables presented in Eq. (2) due to the absence of the MixColumns operation [34]. In standard implementations, there are two strategies for implementing the final round. One method is to define another table for the last round, and the other one is to use the tables presented in Eq. (1) partially. Since the final round comprises only two operations on the state, i.e., SubBytes and ShiftRows, the values of $S(s_{12}^9)$, $S(s_8^9)$, $S(s_4^9)$, and $S(s_0^9)$ can be calculated through access to table T_0 and use of the second element of T_0. Similarly, other bytes can be calculated through access to tables T_1, T_2, and T_3. The implementation of the final round in the OpenSSl library version 1.1.0f uses the second method.

3. Cache template attacks

Cache template attack on the first round of the AES cipher was proposed in [23]. The attack mainly consists of two phases: 1) profiling phase and 2) key exploitation phase. At the profiling phase, dependencies between the processing of the secret key of the AES and specific cache accesses are determined. In addition, the attacker can accurately determine the start and end of the T-table AES in `libcrypto.so` file of the OpenSSL through the cache hit ratio. At the key exploitation phase, the Most Significant Bits (MSBs) of the key of each byte are retrieved. Both phases are briefly elaborated in the following.

3.1. Profiling phase

The profiling phase measures the cache-hit ratios on specific addresses during the execution of the AES. The cache-hit ratios are stored in a matrix called cache template matrix which has one column per encryption and one row per address. In order to compute the *cache template matrix*, first, AES encryption should be performed to encrypt a plaintext where a specific byte is a constant and fixed value, while other bytes can be random arbitrary byte.

If we assume that the size of each cache line is 64 bytes, the upper four bits of k_i can be profiled for each key byte k_i. It is required that 16 addresses be profiled for each key byte k_i. To be specific, to determine the cache template matrix for each value of the key byte k_0, the attacker flushes the content of the first address in the binary file and performs AES with a fixed key k_0. The first byte of the plaintext p_0 is chosen *0x00*, while other bytes of plaintext, i.e., $(p_1, ..., p_{15})$, are randomly chosen. Then, the content of the same address is accessed and the execution time is measured. In case the access time is less than the threshold, it can be interpreted as cache hit with high probability. The attacker performs the process several times and computes the cache-hit ratio on the same address. During the attack process, the cache hit ratio for each address of the binary file `libcrypto.so` is computed. Further, the attack process is repeated for different values $\{0x10, 0x20, ..., 0xF0\}$ of the first byte of plaintext p_0.

Each column vector of the cache template matrix is called a *profile*. We denote the jth column vector of the matrix by $\vec{p_j}$ which is a profile for a constant value $p_0 = 16.j$ where $0 \leq j \leq 15$. Each row represents the address range of the T-table. In other words, each matrix element represents the cache hit ratio for a constant p_0 and a T-table address.

We should remove all rows that contain redundant information from the matrix by pruning the rows with a small difference between the minimum and maximum cache-hit ratios. One should monitor all addresses $a_{64.i}$ in the binary file `libcrypto.so` during the execution of the AES in order to find the start and end of the T-tables in the mentioned file and create a cache template matrix. For this reason, the runtime speed of the profiling phase is slow.

3.2. Exploitation phase

The attacker performs encryption several times for different chosen plaintexts under an unknown key. As a result, 16-byte keys k_i are attacked sequentially where $0 \leq i \leq 15$. For example, to retrieve the upper four bits of k_0, the plaintexts are chosen randomly, except for the four upper bits of p_0 which are fixed to the same chosen value used at the profiling phase.

For all addresses in the cache template matrix resulting from the profiling phase, the cache activity is constantly monitored; hence, cache hit ratio is stored in a vector $\vec{h}$. The attacker computes the similarity between $\vec{h}$ and each profile $\vec{p_j}$ based on the cache template matrix using the mean square error function $S(\vec{h}, \vec{p}_j)$. Assume that for a profile j', $S(\vec{h}, \vec{p}_{j'})$ has the minimum value. Then, we conclude that for the plaintexts with the fixed value $p_0 \in \{0x00, 0x10, 0x20, ..., 0xF0\}$, the corresponding address of T-table is accessed. The address of the T-table that is accessed corresponding to p_0 is determined at the profiling phase. By considering the profile $\vec{p}_{j'}$, we can determine which cache line has the highest cache-hit ratio and, consequently, compute the four most significant bits of $< s_0 >$. Finally, the four most significant bits of k_0 are exploited using Eq. (3).

$$<k_0> = <p_0 \oplus s_0>. \quad (3)$$

The four MSBs for other bytes of the secret key can similarly be retrieved through the aforementioned method.

4. Our attack scenario

4.1. Modified profiling phase

This section presents an effective method for monitoring the addresses of binary file `libcrypto.so` which accelerates the profiling phase in the cache template attack.

As described in Section 3.1, the distance between two addresses in the monitoring step of the profiling phase is considered 64 bytes by Gruss et al. [23]. In the proposed approach, the attacker should consider all addresses in the binary file with the distance of 64 bytes and repeat the described process for each of them [23]. This approach demands a notable amount of time. Although increasing the distance between the addresses during the search step can be an appropriate solution to this problem, it is still challenging since the address line of the first block of T_0 cannot be found in this way. Of note, restricting the distance between the addressed to 64 bytes slows down the speed. Further, increasing the distance leads to missing the start point as 4096 bytes are allocated for saving the lookups. In order to overcome this challenge, a combined approach is suggested.

Our approach consists of two steps. First, we trace addresses where the difference between two consecutive addresses is d bytes in which $d < 4096$ and the process stops when we find an address where the cache hit ratio is large enough. In the second step, we trace the addresses in the backward direction such that the distance between two consecutive addresses is 64 bytes and the process stops when the cache hit ratio is small. Obviously, this approach enjoys one major advantage, that is, the process of finding the address of the first block of T_0 can be accelerated notably.

Assume a cache line with the size of 64 bytes. The process of modified profiling phase for the upper four bits of k_0 is described in Algorithm 1.

First, choose the byte p_0 from the set $\{0x00, 0x10, 0x20, ..., 0xF0\}$ and generate other bytes

```
Input: Binary file $B$, distance $d$ where $d < 4096$
Output: Cache template matrix $M$ for $k_0$
Map libcrypto.so file into memory
foreach $p_0 \in \{0x00, 0x10, 0x20, ..., 0xF0\}$ do
    for $(i = 0, i++, i \leq e/d)$ do
        $a_i = d \times i$
        for $(j = 1, j++, j < N)$ do
            $p_1 \parallel p_2 \parallel ... \parallel p_{15}$=rand()
            P $\longleftarrow (p_0 \parallel p_1 \parallel p_2 \parallel ... \parallel p_{15})$
            Flush $(a_i)$
            $AES_k(P)$
            $t_1 \longleftarrow$ start time
            Reload $(a_i)$
            $t_2 \longleftarrow$ end time
            $t_{p_0} = t_{p_0} + t_2 - t_1$
            $c_{p_0} = c_{p_0} + 1$
            $d_{p_0} = t_{p_0}/c_{p_0}$
            if $(d_{p_0} <$ threshold$)$ then
                $L[p_0][a_i]++$
            end
        end
    end
    if $L[p_0][a_i] \geq N/2$ then
        $a_{start} = a_i$ break
    end
end
foreach $(p_0 \in \{0x00, 0x10, 0x20, ..., 0xF0\})$ do
    for $(i = 0, i++, i \leq e)$ do
        $\hat{a}_i = (a_{start} - d) + 64 \times i$
        for $(j = 1, j++, j \leq N)$ do
            $p_1 \parallel p_2 \parallel ... \parallel p_{15}$=rand()
            P $\longleftarrow (p_0 \parallel p_1 \parallel p_2 \parallel ... \parallel p_{15})$
            Flush $(\hat{a}_i)$
            $AES_k(P)$
            $t_1 \longleftarrow$ start time
            Reload $(a_i)$
            $t_2 \longleftarrow$ end time
            $t_{p_0} = t_{p_0} + t_2 - t_1$
            $c_{p_0} = c_{p_0} + 1$
            $d_{p_0} = t_{p_0}/c_{p_0}$
            if $d_{p_0} >$ threshold then
                $L[p_0][\hat{a}_i]++$
            end
        end
    end
    if $L[p_0][\hat{a}_i] < N/2$ then
        break
    end
    $M[p_0][\hat{a}_i] = L[p_0][\hat{a}_i]$

end
```

Algorithm 1. Profiling phase algorithm for k_0.

of plaintext randomly. Then, consider addresses $a_i = d \times i$ for $0 \leq i \leq e/d$ where e denotes the address of the last byte in the binary file B. We repeat the Flush+Reload process N times for each address a_i and the cache-hit ratio for a_i is computed and saved as $L[p_0][a_i]$. The *threshold* used in Algorithm 1 denotes the minimum cache miss cycles which depends on the processor. Finally, stop the process if the cache-hit ratio for a specific $a_{start} = a_i$ is larger than $N/2$. In the second step, consider the addresses $\hat{a}_i = (a_{start} - 1024)64 \times i$. For each address, repeat the Flush+Reload process N times, compute the cache-hit ratio $L[p_0][\hat{a}_i]$, and construct the cache template matrix $M[p_0][\hat{a}_i]$. Finally, stop the process when the cache-hit ratio for $\hat{a}_i$ is less than $N/2$.

4.2. Experimental results

The main advantage of the proposed method and cache template attack, compared to existing cache attacks against AES [21,22], is that they are fully automated. In addition, the cache template attack requires an extremely less amount of data (only 16-160 encryptions). Once the binary is deployed on the target system, it performs both profiling and exploitation phases automatically and returns the key byte candidates to the attacker. Of note, contrary

Table 1. Comparison results for the runtime speed (second) of the attacks.

Number of the encryption	**Attack [23]**	**Our attack ($d = 1024$)**	**Our attack ($d = 2048$)**	**Our attack ($d = 3072$)**	**Our attack ($d = 4096$)**
10	9.021 s	0.560	0.275	0.189	0.145
50	33.773 s	2.128	1.062	0.711	0.527
100	72.927 s	4.362	2.028	1.354	1.017
500	364.830 s = 6 min	19.650	9.838	6.778	4.924
1000	681.916 s = 10 min	39.343	23.090	13.216	9.832

to the attacks [21,22], the attacker at the profiling phase of the cache template attack does not need prior knowledge about the addresses of the T-table elements and can accurately determine the start and end of the T-table implementation in the binary file.

At the profiling phase proposed in the original paper, the attacker should construct 16 profiles to determine the cache template matrix M one key byte. To construct each profile, Flush+Reload technique is performed for the addresses a_i where $0 \leq i \leq [\frac{e+1}{64}]$ and each time is repeated under N encryptions. Consequently, the time complexity of the original method is $16 \times [\frac{e+1}{64}] \times N$ where e denotes the address of the last byte in the binary file and N denotes the number of required encryptions. In the proposed approach, the time complexity of the profiling phase is approximately $16.\frac{e+1}{d} \times N + 16 \times N$ where d is the distance between two consecutive addresses. The time complexity is dominated by the term $16.\frac{e+1}{d} \times N$. According to our expectation, the proposed approach performs faster than original proposal by a factor of $\frac{d}{64}$ where $d < 4096$.

To verify the viability of the theoretical model, the proposed approach and original method given in [23] were employed. The experimental results are summarized in Table 1. The proposed approach was then tested on the openSSL library (version 1.1.0f) under the Ubuntu 16:04 operating system performed in the Intel Corei5-2.50 GHz.

Different distance values between the two consecutive addresses, denoted by d, were taken into consideration. According to Table 1, the profiling-phase runtime of the cache template attack is around 10 minutes; however, the proposed method speeds up the running of this phase up to approximately 39, 23, 13, and 9 seconds for d = 1024, 2048, 3072, and 4096 bytes, respectively. Therefore, this approach is approximately 16, 32, 48, and 64 times faster than the original proposal for d = 1024, 2048, 3072, and 4096 bytes, respectively.

5. Conclusion

Cache template attack is a method used to automate the process of finding exploitable cache vulnerabilities. In this respect, the current study revisited the cache template attack on the T-table-based implementation of AES and proposed an efficient technique to speed up the profiling phase process. Finally, the proposed approach was employed to experimentally validate the theoretical model. The experimental results confirmed that the given approach was faster than the original method.

Acknowledgments

This study was supported in part by Iran National Science Foundation under Grant 96-53979 and partly by Iran Telecommunication Research Center.

References

1. Buch, D.H. and Bhatt, H.S. "Trinetra: A solution to handle cross-VM time-driven attack", *SN Applied Sciences*, **2**(4), pp. 1–12 (2020).
2. Yarom, Y., Genkin, D., and Heninger, N. "CacheBleed: a timing attack on OpenSSL constant-time RSA", *Journal of Cryptographic Engineering*, **7**(2), pp. 99–112 (2017).
3. Götzfried, J., Eckert, M., Schinzel, S., et al. "Cache attacks on Intel SGX", *Proceedings of the 10th European Workshop on Systems Security*, pp. 1–6 (2017).
4. Xinliang, M., Liehui, J., and Rui, C. "Survey of access-driven cache-based side channel attack", *Journal of Computer Research and Development*, **57**(4), p. 824 (2020).
5. Schwarz, M. "Software-based Side-Channel Attacks and Defenses in Restricted Environments", Doctoral Dissertation, PhD Thesis, Graz University of Technology (2019).
6. Chen, S., Liu, F., Rui, C., et al. "Leveraging hardware transactional memory for cache side-channel defenses", *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 601–608 (2018).
7. Chattopadhyay, S., Beck, M., Rezine, A., et al. "Quantifying the information leakage in cache attacks via symbolic execution", *ACM Transactions on Embedded Computing Systems (TECS)*, **18**(1), pp. 1–27 (2019).
8. Lapid, B. and Wool, A. "Navigating the Samsung trustzone and cache-attacks on the keymaster trust-

let", *European Symposium on Research in Computer Security*, pp. 175–196 (2018).

9. Ge, Q., Yarom, Y., Cock, D., et al. "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware", *Journal of Cryptographic Engineering*, **8**(1), pp. 1–27 (2018).
10. Hu, W.M. "Lattice scheduling and covert channels", *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 52–61 (1992).
11. Kelsey, J., Schneier, B., Wagner, D., et al. "Side channel cryptanalysis of product ciphers", *European Symposium on Research in Computer Security*, pp. 97–110 (1998).
12. Tsunoo, Y., Saito, T., Suzaki, T., et al. "Cryptanalysis of DES implemented on computers with cache", *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 62–76 (2003).
13. Bernstein, D.J., *Cache-timing Attacks on AES* (2005).
14. Aciiçmez, O., Schindler, W., and Koç, Ç.K. "Cache based remote timing attack on the AES", *Cryptographers' Track at the RSA Conference*, pp. 271–286 (2007).
15. Neve, M., Seifert, J., and Wang, Z. "A refined look at Bernstein's AES side-channel analysis", *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, pp. 369–369 (2006).
16. Percival, C. "Cache missing for fun and profit", *BSDCan.* (2006).
17. Yarom, Y. and Falkner, K. "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack", *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pp. 719–732 (2014).
18. Ronen, E., Paterson, K.G., and Shamir, A. "Pseudo constant time implementations of TLS are only pseudo secure", *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.*, pp. 1397–1414 (2018).
19. Ronen, E., Gillham, R., Genkin, D., et al. "The 9 lives of Bleichenbacher's CAT: New cache attacks on TLS implementations", *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 435–452 (2019).
20. Inci, M.S., Gulmezoglu, B., Irazoqui, G., et al. "Cache attacks enable bulk key recovery on the cloud", *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 368–388 (2016).
21. Irazoqui, G., Inci, M.S., Eisenbarth, T., et al. "Wait a minute! A fast, cross-VM attack on AES", *International Workshop on Recent Advances in Intrusion Detection*, pp. 299–319 (2014).
22. Gülmezoğlu, B., Inci, M.S., Irazoqui, G., et al. "A faster and more realistic flush+ reload attack on AES", *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 111–126 (2015).
23. Gruss, D., Spreitzer, R., and Mangard, S. "Cache template attacks: Automating attacks on inclusive last-level caches", *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pp. 897–912 (2015).
24. Ge, Q., Yarom, Y., Li, F., et al. "Contemporary processors are leaky and there is nothing you can do about it", *The Computing Research Repository. arXiv* (2016).
25. Gruss, D., Maurice, C., and Mangard, S. "Rowhammer.js: A remote software-induced fault attack in javascript", *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 300–321 (2016).
26. Irazoqui, G. and Guo, X. "Cache Side Channel Attack: Exploitability and Countermeasures", *Black Hat Asia*, **2017**(3), pp. 1–72 (2017).
27. Saileshwar, G. and Qureshi, M.K., *Lookout for Zombies: Mitigating Flush+ Reload Attack on Shared Caches by Monitoring Invalidated Lines*, arXiv Preprint arXiv:1906.02362 (2017).
28. Lipp, M., Schwarz, M., Gruss, D., et al., *Meltdown*, arXiv Preprint arXiv:1801.01207 (2018).
29. Schwarz, M., Schwarzl, M., Lipp, M., et al. "Netspectre: Read arbitrary memory over network", *European Symposium on Research in Computer Security*, pp. 279–299 (2019).
30. Kocher, P., Horn, J., Fogh, A., et al. "Spectre attacks: Exploiting speculative execution", *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19 (2019).
31. Minkin, M., Moghimi, D., Lipp, M., et al., *Fallout: Reading Kernel Writes from User Space*, arXiv Preprint arXiv:1905.12701 (2019).
32. Seddigh, M. and Soleimany, H. "Enhanced Flush+ Reload Attack on AES", *ISeCure*, **12**(2), pp. 81–89 (2020).
33. Daemen, J. and Rijmen, V., *The Design of Rijndael: AES-the Advanced Encryption Standard*, Springer Science & Business Media (2013).
34. Rebeiro, C., Mukhopadhyay, D., and Bhattacharya, S., *Timing Channels in Cryptography: A Micro-Architectural Perspective*, Springer (2014).

Biographies

Mahdi Esfahani received his PhD degree in Applied Mathematics from Islamic Azad University, Iran in 2020. He has also been working as a researcher at Information Systems and Security Lab (ISSL) at Sharif University of Technology, Iran since 2018. His main research interests are cryptography, timing side-channel attacks, micro-architectural side-channel attacks, and cache-based attacks.

Hadi Soleimany has been working as an Assistant Professor at Cyberspace Research Institute at Shahid Beheshti University, Iran since 2015. He received his PhD in Theoretical Computer Science from Aalto University, Finland in 2015. He was also a postdoctoral researcher at Technical University of Denmark (DTU), Denmark in Summer 2016 and 2017. His main research interests are practical aspects of cryptography.

Mohammad Reza Aref received his BSc degree in 1975 from the University of Tehran, Iran and MSc and PhD degrees in 1976 and 1980, respectively, from Stanford University, Stanford, CA, USA, all in Electrical Engineering. He returned to Iran in 1980 and since then, he has been actively engaged in academic activities. He was a former Faculty member of Isfahan University of Technology from 1982 to 1995. He has been working as a Professor of Electrical Engineering at Sharif University of Technology, Tehran since 1995 and has published more than 230 technical papers in communication and information theory and cryptography in international journals and conferences proceedings. His current research interests include areas of communication theory, information theory, and cryptography.