DOI: 10.21917/ijme.2023.0258

DETECTING THE SIDE CHANNEL ATTACK IN EMBEDDED PROCESSORS USING FEDERATED MODEL

J. Seetha[1], Ananda Ravuri[2], Yamini Tondepu[3] and T. Kuntavai[4]

[1]*Department of Computer Science and Business Systems, Panimalar Engineering College, India*
[2]*Software Engineer, Intel Corporation, Oregon, United States*
[3]*Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, India*
[4]*Department of Electrical and Electronics Engineering, Adhiparasakthi Engineering College, India*

Abstract

Embedded processors, which are an essential component of any Internet of Things (IoT) device, are vulnerable to a considerable risk posed by power side-channel attacks. Defences against power attacks have been implemented at the application level, such as ways for masking a device power use, or at the hardware level, in the form of leakage concealment methods. In this paper, we describe an innovative method for mitigating power side channel attacks through the utilisation of integrated cache. We employ an open-source embedded cache simulator that can be used as the basis for the embedded cache model. The results show that Federated Learning (FL) can discover attack sequences on real-world processors without requiring knowledge of the processor replacement policies and prefetchers.

Keywords:

Attack, Federated Learning, IoT, Processor, Embedded

1. INTRODUCTION

Embedded processors, which are an essential component of any Internet of Things (IoT) device, are vulnerable to a considerable risk posed by power side-channel attacks [1]. These types of attacks can obtain sensitive information that has been exposed because they exploit a relationship that exists between the sensitive data and the amount of power that the processor uses [2]. These methods have seen considerable application in recent years for the purpose of deciphering encryption algorithms and recovering misplaced keys [3].

Defences against power attacks have been implemented at the application level, such as ways for masking a device power use [4], or at the hardware level, in the form of leakage concealment methods [5]. However, these approaches result in enormous additional expenses because of the enormous increases in area and energy required to carry them out. This means that they can only be utilised in very specialised places, such as cryptography modules, and that the rest of the system is wide open to attack because of it [6].

To protect the CPU from power vulnerabilities, we investigate the primary cause of the leakage and then implement remedies for it, module by module. This is performed by verifying that the translated EDA not only operates as expected but also does not introduce any extra side-channel vulnerabilities into the system. In addition to this, we make it a point to hide the data path of the CPU as much as possible.

This indicates that the data that is saved in locations such as the pipeline buffers, line buffers, GPRs, and internal cache memory of the CPU is all encrypted. Because the obfuscation eliminates any possible link between the data and the amount of power consumed, the leakage has been significantly reduced. Only in the execution stage of the processor does data de-obfuscation take place when it is necessary to perform an operation on the data. Before it is written to the registers, the result of the operation is concealed once more [7].

Embedded caches are susceptible to three distinct types of side-channel attacks, each of which can be launched by adversaries possessing a different level of capability:

- *Access-driven SCA*: This is a type of attack that an adversary can employ to identify what data sets of embedded cache a victim has accessed in the past. The Prime and Probe attack is the most well-known strategy that can be used in this scenario. A new prime and probe attack has also been developed by our team. This attack is a variation on the one that was developed previously, and it involves the disclosure of private data via the exchange of memory pages and embedded CPU cache lines. The spy process analyses the amount of time necessary to get certain lines of code from the victim shared memory and compares the results. Also, there are many different varieties of access-driven embedded cache side-channel attacks to choose from [12].
- *Time-driven SCA*: This is an attempt to calculate the total amount of time necessary to carry out a sequence of cryptographic procedures using a given key. These attacks are carried out by computers. The value of the key influences the total amount of time it takes for the programme to run. For the attacker to determine whether the victim process makes use of a specific embedded cache set, they will cause some interruption [13].
- *Trace-driven SCA*: The total number of misses and hits that occur in the embedded cache of the machine or process that is being targeted [14].

In this research, we describe an innovative method for mitigating side-channel attacks through the utilisation of integrated cache. An approach to detection that guards cloud servers from embedded cache side-channel attacks This study focuses on the hardware that is used to host cloud computing tenants. This is accomplished by tracking failures of embedded caches that are caused by embedded CPUs.

Additionally, the frequency with which embedded cache misses occur, as well as the circumstances under which they occur, will be tracked, and examined in this study. The study constructs a sequence, a standard deviation would be calculated, and then that number would be compared to a threshold.

2. RELATED WORK

There is a detective solution that are capable of being implemented in the cloud as a direct result of multi-tenancy and

co-residency. The HomeAlone solution was developed by [8] to provide tenants of cloud computing with the ability to ensure that they are physically segregated from one another and to identify the co-resident harmful embedded processor. Embedded cache side-channel analysis is a sort of defensive detection that tenants of the cloud are now able to use.

This detection method is only applicable to the one case in which the two cloud tenants are geographically isolated from one another. To determining whether the integrated cache in the CPU is behaving normally, a classifier is used. Within a tenancy, it is needed that each embedded processor has a coordinator and remapper implemented so that it can store information in preset embedded cache sets and share information with other embedded processors. It possible that the presence of another embedded processor will require virtual machines to be shut down as a result.

Our method, on the other hand, does not require the utilisation of a classifier tool, any modifications to the guest operating systems or kernels, or the muting of any embedded processors. It does not make things go more slowly, and it does not bring any more stress. The CPU integrated cache at all levels is inspected, and any malicious integrated processor is discovered and differentiated from other embedded processors in the system.

In [9] suggests using a two-stage mode approach to detect embedded cache-side channel attacks that take place in the cloud. The procedure consists of two stages. The OProfile can generate at least 2,000 interrupts per second [10] for the CPU. These interrupts can be used to measure the number of times an embedded processor causes an embedded cache to miss on the CPU.

For an embedded processor to interface with its host and obtain data, it must additionally have an agent built into it. In addition, there is a high percentage of false negatives, which is 40% [11]. On the other hand, the approach that we offer does not require the use of any additional software or mobile applications. The number of cache misses that occur during the fetch cycle of the CPU can be easily quantified. In addition, the hypervisor kernel only consists of a single stage of the implementation process.

3. SIDE-CHANNEL FACTOR

The side-channel vulnerability factor (SVF) is a statistic that can be used to determine how susceptible a device is to the leakage of information through side channels. It determines how closely the execution pattern of a sensitive application matches up with the side-channel observations of an attacker. If the correlation is high, then there must be a significant amount of leakage across the side channel.

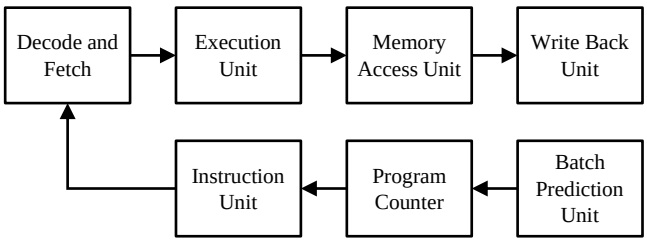

Fig.1. Main Units of Embedded Processors

When computing SVF, there are two stages: the online stage and the post-processing step. In the context of the online environment, N different stimuli are used. Each input has a ground truth that is constructed in an Oracle trace and is associated with it. This ground truth is something that an enemy would want to read from the device.

$$O = (o_1, o_2, \ldots, o_i, \ldots .o_N), \quad (1)$$

where the value of the truth for the i^{th} input, indicated by o_i ($1 \le i \le N$), can be found. In the case of an AES victim, for example, the execution is initiated by changing the input or the key. The value $S(p \oplus k)$ might be an illustration of an AES S-box operation, and it could appear in the Oracle Trace. In this case, p would be a plaintext byte, k would be the associated secret key byte, and S would be the AES S-box operation. A side channel trace that contains information on the device power usage trends is prepared at the conclusion of each run.

$$S = (s_1, s_2, ,..., s_j ,... s_N) \quad (2)$$

where s_j represents the power curve as measured at input j in the equation. By the time the online portion of the study was over, two separate collections of data had been obtained. Following the completion of the collection of all the traces, the data are analysed to identify patterns within the traces and determine how they are connected. To uncover patterns, a distance metric is applied to each trace data to calculate the pairwise distances between the data.

$$D_O = \text{distance}(o_i,o_j), \forall o_i,o_j \in O \text{ and } i > j. \quad (3)$$

In a similar manner, a distance metric is applied to S to construct a side-channel distance vector DS. The next step in determining the SVF involves computing the Pearson correlation that exists between these two vectors. If the SVF is set too high, the Oracle suffers from severe side-channel leakage. The distance measurement that is used will be decided based on the data that is submitted. In this investigation, we use the Hamming distance metric to solve the challenge of determining how far apart binary data sets are from one another. The following is our working definition of the Hamming distance:

$$\text{HD}(x_i, x_j) = (n\text{-}1)^b = 0 \text{ where } h_b \in N, \quad (4)$$

where x_i and x_j are binary strings with length n, and h_b is the bit that corresponds to the b^{th} position in the product of $(x_i \oplus x_j)$.

4. SCA DETECTION

We begin our search for components that leak side-channel information with the RTL of the CPU. Once we locate these components, we patch them so that the processor is more resistant to side-channel attacks. After that, we implemented fixes for the modules that were causing the problems. By studying the RTL of the processor modules, which we publish here, our approach could identify leaky processor modules. A list of software components is used to model the central processing unit.

Each module has its own unique collection of signals and submodules, all of which are connected to one another through a complex web of wires and registers. This method conducts a separate analysis of each individual component of the CPU. It determines the quantity of signal that is lost due to leakage from the module. Leakage of individual sub-modules is evaluated separately. The formal definition of the processor netlist, denoted

by P, is the collection of all the components, also known as modules.

$$P = \{M_1, M_2, ..., M_i, ..., M_m\} \quad (5)$$

Each module comprises of signals (wires and registers), which we denote by the set. $M_i (1 \leq i \leq m)$ is the i^{th}. The term set refers to the components of a module, which include wires and registers, respectively.

$$M_i = \{S_1(i), S_2(i), ..., S_j(i), ..., S_n(i)\}, \quad (6)$$

where $S_j(i)$ $(1 \leq j \leq n)$ is a signal that might have one bit or multiple bits and is specified by the module M_i. The creation is based on the following set of assumptions:

The amount of energy required to process a k-bit signal S is directly proportional to its Hamming weight. It is being referred to by the b^{th} bit of S. Define the function $Y(M_i)$ so that it is the union of all the signals that are contained in M_i. This function can be described in the following manner by us:

$$Y(M_i) = (S_1(i) \,||\, S_2(i) \,||\, ... \,||\, S_n(i)) \quad (7)$$

The concatenation operator is represented by ||. Therefore, if the bit width of each M_i signal is 32, then the bit width of $Y(M_i)$ will be $32 \times n$ if we assume that each M_i signal has its own bit width. The predicted overall power consumption of Module M_i is obtained by aggregating the individual power requirements of each of its signals. This indicates that the amount of information kept in $Y(M_i)$ is directly proportional to the amount of energy that module M_i consumes. Calculating the SVF between $Y(M_i)$ and the secret data can provide one with an estimate of the amount of side-channel leakage that is caused by the module M_i.

Running benchmark routines multiple times with different inputs while using a netlist that has already been synthesised. Before beginning the actual execution, the user will go through each benchmark programme and select one or more of the exciting phases. Some of the more intriguing operations included in the first phase of an AES benchmark programme.

These are the locations where Oracle trace data is saved. The information included in $Y(M_i)$ is what is used to build the side-channel trace, which can then be used to measure the amount of side-channel leakage coming from a module called M_i. $Y(M_i)$ is a time-series vector, and the members of this vector represent the data that is present during each clock cycle for each benchmark run.

5. FEDERATED LEARNING

The difficulty of federated learning is illustrated in Fig.1 and consists of the following: training a high-quality shared global model on a centralised server using data that is spread among many clients. Let imagine, for the purpose of argument, that we have K clients (in this context, anything from a smartphone to a smartwatch to the data warehouse at a hospital counts as a client).

We refer to the data distribution of the customer K as D_k, and the sample size of the customer K will be referred to as n_k. To get an accurate total, apply the formula $n=\sum Kk=1nk$. The topic of federated learning can be boiled down to the following when an empirical risk minimization problem of this kind is considered:

$$\min_{w \in R} dF(\mathbf{w}) := \sum_k nknFk(\mathbf{w}) \quad (8)$$

$$\text{where} \quad Fk(\mathbf{w}) := (nk)^{-1} \sum_{\mathbf{x}i \in D} kfi(\mathbf{w}),$$

where $\mathbf{w}$ represents the undetermined value of the model learning parameter. The loss function that is used to specify the function fi is defined with the help of the input-output data pair known as $\{\mathbf{x}_i, y_i\}$. In most cases, $\mathbf{x}_i \in R^d$ and $y_i \in R$ or $y_i \in \{-1, 1\}$ are used. Some obvious examples to consider are the following:

$$f_i(\mathbf{w}) = 0.5(\mathbf{x}_i^T \mathbf{w} - y_i)^2, \; y_i \in R; \quad (9)$$

$$fi(\mathbf{w}) = -\log(1+\exp(-y_i \, \mathbf{x}_i^T \mathbf{w})), \; y_i \in \{-1,1\}; \quad (10)$$

$$fi(\mathbf{w}) = \max\{0, 1-y_i \mathbf{x}_i^T \mathbf{w}\}, \; y_i \in \{-1,1\}. \quad (11)$$

When attempting to launch a side-channel attack on an embedded CPU, there are several challenges that must be surmounted.

5.1 STATISTICAL CHALLENGE

Due to the fact that customer data is dispersed in a very non-uniform manner $\forall k \neq k\sim$: $E\mathbf{x}_i \sim D_k[f_i(\mathbf{w};\mathbf{x}_i)] \neq E\mathbf{x}_i \sim D_k\sim[f_i(\mathbf{w};\mathbf{x}_i)]$ $F(w)$ is defined in such a way that any locally available data points are not even close to being able to be considered a representative sample of the overall distribution.

$$E\mathbf{x}_i \sim D_k[f_i(\mathbf{w};\mathbf{x}_i)] \neq F(\mathbf{w}). \quad (12)$$

5.2 COMMUNICATION EFFICIENCY

The number of customers K is substantial, and it is possible that it is much higher than the average number of training samples retained by active consumers, n, i.e., $K \gg (n/K)$.

5.3 PRIVACY AND SECURITY

Additional precautions must be taken to protect client privacy who are deemed to be untrustworthy parties. It is not reasonable to think that every single customer can be relied upon.

6. EVALUATION

The embedded cache simulator makes testing for potential vulnerabilities in both new and older embedded cache systems much more efficient. Exploring attacks on actual hardware makes it possible to apply FL to real-world system designs, even when the design details of those system designs are unknown. This is because actual hardware is easier to exploit. An open-source embedded cache simulator that can be used as the basis for the embedded cache model has been incorporated into the FL framework.

By making a few adjustments to the simulator for embedded caches, it is possible to create multi-level embedded caches. We have been employing embedded caches that are physically indexed as well as physically tagged to keep things as simple as possible, and we have also been allowing both the attacker programmes and the victim programmes to make direct use of physical addresses when making requests. This has enabled us to keep things as plain as possible.

Table.1. Comparison of Various Schemes

Scheme	Number of Multipliers (G)	Top-5 Error (%)	Speed upto	Fine-Tuning Time (epochs)
HomeAlone	15.49	9.52	1.23×	5-10
Remapper	7.70	10.23	2.2×	10
CPU integrated cache	4.56	13.16	3.12×	15
OProfile	3.85	9.52	4.23×	1-6
Hypervisor kernel	3.04	11.02	5.11×	2-10
Proposed	1.32	9.62	10.23×	1-5

Table.3. Comparison with Various Attack Modes

Device	Attack Mode	HomeAlone	Remapper	CPU integrated cache	OProfile	Hypervisor kernel	Proposed
Device 1	Unauthorized Access	26.95	53.90	105.56	210.00	421.13	844.50
	DDoS	26.95	62.89	106.69	214.49	430.11	862.46
	Man in the Middle	15.72	32.57	64.01	129.15	258.29	518.83
	Insider threats	15.72	37.06	65.13	131.39	262.78	527.81
Device 2	Unauthorized Access	553.64	1107.28	2213.43	4423.50	8858.22	17766.98
	DDoS	552.52	966.90	1693.48	2960.23	5193.88	9475.87
	Man in the Middle	335.78	671.55	1343.11	2690.71	5375.80	11157.01
	Insider threats	335.78	609.79	1114.02	2041.61	3759.80	7263.56
Device 3	Unauthorized Access	275.14	550.27	1100.54	2203.33	4397.67	8824.53
	DDoS	277.38	486.26	852.36	1491.34	2610.98	4577.35
	Man in the Middle	143.74	287.49	576.10	1151.08	2301.03	4606.55
	Insider threats	143.74	257.17	461.55	827.65	1499.21	2714.29
Device 4	Unauthorized Access	548.02	1098.29	2194.34	4388.68	8780.74	17562.60
	DDoS	554.76	970.27	1699.10	2973.70	5195.00	9141.22
	Man in the Middle	285.24	571.61	1142.09	2283.06	4576.23	9172.66
	Insider threats	285.24	510.97	914.12	1646.32	2970.34	5402.75
Device 5	Unauthorized Access	280.75	563.75	1130.86	2250.49	4505.48	9069.35
	DDoS	278.50	489.63	854.60	1491.34	2601.99	4579.59
	Man in the Middle	132.51	265.03	528.93	1057.87	2112.36	4240.45
	Insider threats	132.51	234.71	415.51	736.69	1310.54	2341.46
Device 6	Unauthorized Access	559.25	1124.12	2243.75	4486.39	8925.60	17846.72
	DDoS	554.76	974.76	1692.36	2971.46	5214.09	9098.55
	Man in the Middle	259.41	518.83	1036.53	2070.81	4147.24	8289.99
	Insider threats	259.41	460.43	814.18	1444.18	2578.41	4591.95
Device 7	Unauthorized Access	274.01	546.90	1100.54	2178.62	4390.93	8781.86
	DDoS	270.64	472.78	829.90	1440.81	2526.75	4448.20
	Man in the Middle	130.27	259.41	517.70	1033.16	2065.20	4132.64
	Insider threats	130.27	229.09	402.03	707.49	1247.65	2202.20
Device 8	Unauthorized Access	537.92	1072.47	2152.79	4304.46	8619.03	17203.24
	DDoS	532.30	930.97	1631.72	2859.16	4993.98	8771.75
	Man in the Middle	249.31	497.49	992.73	1983.22	3957.45	7937.36
	Insider threats	250.43	439.09	770.38	1358.83	2390.87	4235.96
Device 9	Unauthorized Access	132.51	266.15	532.30	1067.97	2133.70	4293.23
	DDoS	131.39	231.34	406.53	708.61	1243.16	2175.25
	Man in the Middle	73.00	143.74	285.24	569.36	1133.11	2257.23
	Insider threats	73.00	126.90	221.23	387.44	680.54	1193.75

It is time-consuming to have the agent interface with hardware for each operation, and doing so when working with actual hardware makes the training more susceptible to disruption from system noise. When training on real hardware, we tackle this problem by performing all the instructions contained in each episode simultaneously. All the instructions contained in an episode are carried out; however, the agent is only able to determine the latency of memory accesses after they have made a guess.

Unless otherwise specified, all training is executed on clusters that are equipped with Intel 2.20 GHz CPUs. The hardware that is being used in the following table is representative of actual hardware tests.

7. DISCUSSION

The results show that FL can discover attack sequences on real-world processors without requiring knowledge of the processor replacement policies, prefetchers, or anything else of the sort. In most cases, human experts are required to have access to such details to move recognised dangers to a new platform. For instance, efficiently priming and probing an embedded cache set requires knowledge of the replacement policy. This can be a challenge. On the other hand, since replacement policies for modern CPUs are rarely defined openly by the vendor, it can be challenging to precisely reverse engineer these rules.

After manually reverse engineering an undetermined replacement method from a real-world processor for a significant period, it is possible to manually generate attack sequences. This can be done in any order. According to the findings of our investigations, FL can locate potentially useful attack sequences in a matter of hours.

Because of the versatility of the embedded cache simulator, we can study a far wider variety of embedded cache and attack scenarios with much less effort. To determining how well it will perform in a range of contexts, we conducted FL testing using a wide variety of embedded cache and attack/victim programme combinations. In these kinds of configurations, the LRU replacement policy is always put into effect. The structure of the attack and victim programming place restrictions on the many kinds of attacks that can be carried out. We discovered that the FL agent was able to identify potentially successful attack sequences regardless of the environment.

This is since it may be impossible to attain a global optimum in complicated combinations. However, the training will eventually converge to local optima, which capture the important mechanism that allows the attack for each configuration but have a longer route. It is possible that the agent will uncover a particularly fascinating pattern of attacks.

In contrast to deterministic replacement policies, where future states are entirely predictable given the action and the current state, pseudorandom replacement policies make it difficult to forecast future states. When used in one evaluation, a successful attack sequence may lead to an inaccurate assessment in a different evaluation. The FL agent is also capable of generating several answers, each of which is customised to the data that is currently available. Given that the eviction rate in a random replacement policy is dependent on both the total number and the timing of memory accesses, it follows that no single attack sequence will always succeed against such a strategy. This is because the eviction rate is dependent on both the total number and the timing of memory accesses. Instead, we put the FL agent through a total of one hundred different tests to determine how effective it is as an attacking agent.

8. CONCLUSION

Our tests have shown that the FL is capable of efficiently discovering attack sequences against a wide variety of embedded cache implementations. The FL agent was also responsible for discovering this novel attack, which had a greater bit rate on actual systems when compared to attacks recorded in the older literature. When it comes to analysing timing attacks on microarchitectures in real-world systems, FL has proven to be an extremely helpful tool.

REFERENCES

[1] Y. Guo and J. Yang, "Adversarial Prefetch: New Cross-Core Cache Side Channel Attacks", *Proceedings of IEEE Symposium on Security and Privacy*, pp. 1458-1473, 2022.

[2] J. Wan and Z. Li, "MeshUp: Stateless Cache Side-Channel Attack on CPU Mesh", *Proceedings of IEEE Symposium on Security and Privacy*, pp. 1506-1524, 2022.

[3] I. Buhan and P. Schaumont, "SoK: Design Tools for Side-Channel-Aware Implementations", *Proceedings of ACM on Asia Conference on Computer and Communications Security*, pp. 756-770, 2022.

[4] R. Kumar, M.A. Anders and S.K. Mathew, "A Time-/Frequency-Domain Side-Channel Attack Resistant AES-128 and RSA-4K Crypto-Processor in 14-nm CMOS", *IEEE Journal of Solid-State Circuits*, Vol. 56, No. 4, pp. 1141-1151, 2021.

[5] A. Sayakkara and M. Scanlon, "A Survey of Electromagnetic Side-Channel Attacks and Discussion on their Case-Progressing Potential for Digital Forensics", *Digital Investigation*, Vol. 29, pp. 43-54, 2019.

[6] M. Mushtaq, M.K. Bhatti and G. Gogniat, "Winter is Here! A Decade of Cache-based Side-Channel Attacks, Detection & Mitigation for RSA", *Information Systems*, Vol. 92, pp. 101524-101534, 2020.

[7] N. Tsalis and T. Apostolopoulos, "A Taxonomy of Side Channel Attacks on Critical Infrastructures and Relevant Systems", *Proceedings of International Conference on Critical Infrastructure Security and Resilience: Theories, Methods, Tools and Technologies*, pp. 283-313, 2019.

[8] M. Rajalakshmi and C. Karthik, "Machine Learning for Modeling and Control of Industrial Clarifier Process", *Intelligent Automation and Soft Computing*, Vol. 32, No. 1, pp. 1-14, 2022.

[9] Y. Guo and J. Yang, "Adversarial Prefetch: New Cross-Core Cache Side Channel Attacks", *Proceedings of IEEE Symposium on Security and Privacy*, pp. 1458-1473, 2022.

[10] J. Wan and Z. Li, "MeshUp: Stateless Cache Side-Channel Attack on CPU Mesh", *Proceedings of IEEE Symposium on Security and Privacy*, pp. 1506-1524, 2022.

[11] I. Buhan and P. Schaumont, "SoK: Design Tools for Side-Channel-Aware Implementations", *Proceedings of ACM on*

Asia Conference on Computer and Communications Security, pp. 756-770, 2022.

[12] A. Sayakkara and M. Scanlon, "A Survey of Electromagnetic Side-Channel Attacks and Discussion on their Case-Progressing Potential for Digital Forensics", *Digital Investigation*, Vol. 29, pp. 43-54, 2019.

[13] D. Townley and D. Ponomarev, "SMT-COP: Defeating Side-Channel Attacks on Execution Units in SMT Processors", *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pp. 43-54, 2019.

[14] M. Mushtaq, M.K. Bhatti and G. Gogniat, "Winter is Here! A Decade of Cache-Based Side-Channel Attacks, Detection and Mitigation for RSA", *Information Systems*, Vol. 92, pp. 101524-101533, 2020.