Intel Corporate Research Council



**Intel Corporate Research Council/ University Research Office (URO)**

**GRANT PROPOSAL COVERSHEET & PRIVACY POLICY NOTICE**

Intel is committed to respecting your privacy. The information you provide will be used and retained for processing and funding your grant/gift, for Intel’s audit purposes, and for grant-related correspondence. For more information regarding Intel’s personal information handling practices, please visit [www.Intel.com/Privacy.](http://www.intel.com/Privacy)

**(For grants other than conference sponsorships)** Check here to certify that you are not engaged in prior agreements (e.g. government contracts) that constraint your ability to negotiate mutually agreeable intellectual property terms for this grant



**(For grants other than conference sponsorships)** Check here to notify us that you are engaged in prior agreements (e.g. government contracts) that constrain your ability to negotiate mutually agreeable intellectual property terms for this grant



Check here to opt in for occasional URO updates - announcements or newsletters



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Proposal Title:** | | |
|  | | | | |
| **University Name / Receiving Organization** | |  | **Department/Discipline** | |
|  |  |  |  |  |
| **Representative Authorized**  **To Conduct Grant Administration** | | | **Principal Investigator Information** | |
| **Contact Name** |  |  | **PI Name** |  |
|  |  |  |  |  |
| **Mailing**  **Address** |  | | **Mailing Address** |  |
| **Phone #** |  |  | **Phone #** |  |
| **Fax #** |  |  | **Fax #** |  |
| **E-Mail**  **Address** |  | | **E-Mail Address** |  |
|  |  |  | **Project /PI URL** |  |
|  |  | **Co-Investigator/Students** |  |
| **Amount of**  **Cash**  **Requested** |  | |  |  |
| **Additional** |  |  |  |  |
| **Comments:** |  |  |  |  |
|  |  |  |  |  |

1. **Technical Proposal**
   1. **Introduction and Motivation**

Many modern cryptosystems are vulnerable to side-channel attacks even after the recent advancement of cryptographic algorithms. These attacks compromise the secret key by the information gained through the physical implementation of the cryptosystem. One category of such attack includes Micro-architectural side-channel-attacks which can retrieve the secret key by observing micro-architectural functionalities of the processor implementation, like cache accesses, branch instructions, etc. [2, 3]. Modern microprocessors contain a set of special purpose registers to measure hardware related activities known as hardware performance counters, which leak valuable information regarding the encryption algorithm [1]. Some attacks [4] analyze these performance counters for compromising the security of the system.

There are many state-of-the-art countermeasures to prevent the side-channel attacks, but with the cost of an extra overhead of implementation. Implementation of these countermeasures are not feasible in resource constraint environment like IoT devices, Smart-phones etc. There has been some work [5, 6] to prevent these types of attacks by observing the performance counters of the adversary program, which it leaves behind while executing on the system. One possible way to detect and prevent these attacks is to analyze the hardware footprints of the system in real time using data analytics techniques and classify the state of the system as safe or unsafe. A safe state is when there is no existence of any side-channel-attack, and the unsafe state signifies the execution of any adversary program in the background. Most of these techniques state about preventing the cached-based side-channel attacks using some machine learning methods without generalizing the detection method. Moreover, some advanced techniques like drammer [7] exploit the row hammer hardware vulnerabilities on Android and iOS systems to take full control over the system. This is also a motivation to implement a detection method with negligible implementation overhead such that it can be used in any device with resource constraint.

Apart from side-channel-attacks there are some advanced malwares like FireEye which disrupts the normal flow of a system and gain authorized access and there are also malwares like ransomware, which encrypts victim’s data until a ransom is paid. We believe these kind of malwares also leave behind their executing footprints on the hardware events. This is also a motivation to implement a detection technique which will detect and prevent these types of attacks.

We already started working on a generalized detection approach for micro-architectural side-channel attacks by utilizing the hardware performance counters on both Intel and AMD based platforms [8]. Our objective is to learn a classifier with the behaviors of different anomalies (both attack process and non-attack process with similar micro-architectural behavior) in the target system. The classifier produces a warning to the user about the existence of a possible side-channel attack. We deal with the false positives by correlating the micro-architectural event trace of the abnormal process with the secret key of encryption. Process having a high correlation value is termed as a side-channel attack.

* 1. **Research Plan**

There may be *“intelligent”* adversaries which perform some random operations in between their work-flow to confuse the detection mechanism. Even the works presented in [2, 5] reported the inability to detect these attacks with their detection scheme, i.e., there is chance of false negatives considering these types of attacks.

The execution of an “intelligent” adversary consists of attack part added with some random noise part. These adversaries need to execute some particular portion of their code (i.e., the portion responsible for retrieving the secret key) every time to retrieve the secret key successfully. The hardware trace for this part remains same every time of their execution. In the presence of ’forcefully incorporated random code,’ the hardware trace of the actual attack portion will not be affected, rather will be shifted by some interval in the time domain. The Fig.1 gives a basic sketch of this idea for a simulated example.

Fig. 1: Work-flow of an “intelligent” adversary

Each square in Fig. 1 represents the execution trace of the adversary for one complete iteration of an encryption algorithm. Typical adversary program accesses the encryption process multiple times in continuation to guess the correct secret key. We have handled these types in our current work. However, an “intelligent” adversary will try to fool the detection scheme by introducing some random operations in between each iteration. These random operations will leave their effects on hardware performance counters, but will not hamper the actual attack portion, as this is necessary for the attacker to mount a successful attack.

We believe all the attack processes have these types of ‘special’ portions in their execution. If we can learn these parts of their executions, then we will be able to detect these attacks even without correlating the traces with the secret key, which is much harder in the presence of system noise.

The execution trace of any hardware event for any adversary process follows a particular temporal pattern as the adversary needs to access the encryption algorithm multiple times to get any information about the secret key as shown in the above figure. These temporal patterns could be learned using Long Short- Term Memory (LSTM) Recurrent Neural Networks [3, 4].

Use of LSTM Networks

LSTMs are special kinds of RNNs capable of learning long-term dependencies in a sequence. The advantage of LSTMs are that they are very generic and does not require any feature engineering beforehand, i.e., we need to select any important feature for the decision. We know, the hardware trace of an adversary process is a temporal sequence of different hardware events and to become successful the adversary needs to execute the ‘attack’ portion of the code. Thus, in the presence of a random noise for a random interval of time there exist a dependency in the sequence of values of the hardware events for the adversary process to become a successful attack. This, we believe, is a perfect scenario for the application of an LSTM networks.

* 1. **Demonstration and experimentation plan**

We may approach this problem in the following ways:

1. We don’t have access to the “intelligent” attack codes, but we do have access to most of the typical micro-architectural attack codes. We can incorporate random operations in between them and see whether they are retrieving the correct secret keys or not. If they become successful in recovering the secret key, we can use them as our test cases.

2. Once the test cases are formed, we can use our current detection mechanism and see whether they are detected or not. (I believe they will not be detected with high amount of random operations in the presence of system noise, but I need to check).

3. If we can detect these modified attack codes with the current approach, we can add more randomness without hampering its successful retrieval of keys. Otherwise, we can generate the hardware traces of these attack processes and learn an LSTM network with them.

4. Since we are modifying the attack codes by ourselves, we know which por- tion of the code is actual attack code and which are random noise. This information will help the LSTM network to learn the behavior accurately.

There are two forms of noise: one coming from other system programs, and another coming from introduction of different “intelligent codes”, in the attack code itself. We can generate multiple instances of attack traces incorporating a combination of these types of attacks. For example, for intelligent codes, we can add random sleeps, or random memory reads of large chunks so that they affect cache, etc. similarly we can run different background programs, e.g. browsers, video encoders, compression algorithms, data processing tools, etc to generate different system noises. Our task is to classify all these noisy attack traces as positive, when compared to non-attack traces. We can take some of these noisy attack traces as training data and the rest as test data, and try to classify using LSTM, vs the existing methods.

Cache Miss trace of AES encryption for a particular secret key. Cache Miss trace of conventional cache-timing attack on AES Cache Miss trace of modified intelligent attack on AES

Fig. 2: Temporal Sequences of different scenarios in the absence of background noise.

Cache Miss trace of AES encryption for a particular secret key. Cache Miss trace of conventional cache-timing attack on AES Cache Miss trace of modified intelligent attack on AES

Fig. 3: Temporal Sequences of different scenarios in the presence of background noise.

1. **Organizational Remarks**
   1. **Statement of work, schedule, milestones, deliverables**
   2. **Proposal team**

All the team members for this projects belong to the Department of Computer Science and Engineering at Indian Institute of Technology, Kharagpur.

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | Manaar Alam | : | A 2nd year Ph.D. student. |
| 2. | Sourangshu Bhattacharya | : | Assistant Professor. |
| 3. | Debdeep Mukhopadhyay | : | Associate Professor. Recipient of prestigious Swarnajayanti Fellowship. |

* 1. **Cost volume**

**References**

1. Leif Uhsadel; Andy Georges; Ingrid Verbauwhede. Exploiting Hardware Performance Counters. 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, August 2008.
2. J. Bonneau and I. Mironov. Cache-Collision Timing Attacks Against AES. In L. Goubin and M. Matsui, editors, CHES, volume 4249 of Lecture Notes in Computer Science, pages 201-215. Springer, 2006.
3. Onur AcıiçmezÇetin Kaya KoçJean-Pierre Seifert. Predicting Secret Keys Via Branch Prediction. Topics in Cryptology -- CT-RSA 2007: The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA. February, 2007.
4. Sarani Bhattacharya and Debdeep Mukhopadhyay. Who Watches the Watchmen?: Utilizing Performance Monitors for Compromising Keys of RSA on Intel Platforms. Cryptographic Hardware and Embedded Systems -- CHES 2015: 17th International Workshop, Saint-Malo, France. September, 2015
5. Tianwei Zhang; Yinqian Zhang; Ruby B. Lee. CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds. Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France. September, 2016.
6. Marco Chiappetta; Erkay Savas; Cemal Yilmaz. Real time detection of cache-based side-channel attacks using hardware performance counters. Applied Soft Computing, Volume 49, December 2016.
7. V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, C. Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms, in: CCS, 2016.
8. Alam, M., Bhattacharya, S., Mukhopadhyay, D., Bhattacharya, S.: Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel attacks. IACR Cryptology ePrint Archive 2017(2017) 564.