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:** | | |
| **Detection and Prevention of “intelligent” side-channel adversaries using LSTM network** | | | | |
| **University Name / Receiving Organization** | |  | **Department/Discipline** | |
| **Indian Institute of Technology Kharagpur, India** | | | **Department of Computer Science and Engineering** | |
| **Representative Authorized**  **To Conduct Grant Administration** | | | **Principal Investigator Information** | |
| **Contact Name** | Dr. Pallab Dasgupta |  | **PI Name** | Dr. Sourangshu Bhattacharya |
|  |
| **Mailing**  **Address** | Dean,  Sponsored Research and Industrial Consultancy (SRIC),  IIT Kharagpur,  Kharagpur - 721302,  West Bengal, India | | **Mailing**  **Address** | Department of Computer Science and Engineering,  IIT Kharagpur,  Kharagpur – 721302,  West Bengal, India |
| **Phone #** | +91-3222- 282037 |  | **Phone #** | +91-3222-282364 |
| **Fax #** |  |  | **Fax #** |  |
| **E-Mail**  **Address** | [deansr@hijli.iitkgp.ernet.in](mailto:deansr@hijli.iitkgp.ernet.in)  [pallab@cse.iitkgp.ernet.in](mailto:pallab@cse.iitkgp.ernet.in) | | **E-Mail**  **Address** | [sourangshu@cse.iitkgp.ernet.in](mailto:sourangshu@cse.iitkgp.ernet.in) |
|  |  |  | **Project /PI URL** | <http://cse.iitkgp.ac.in/~sourangshu/> |
|  |  | **Co-Investigator**  **/Students** | Co-PI: Dr. Debdeep Mukhopadhyay  Student: Manaar Alam (Ph.D. Scholar)  Student: Sarani Bhattacharya (Ph.D. Scholar) |
| **Amount of**  **Cash**  **Requested** |  | |  |  |
| **Additional** |  |  |  |  |
| **Comments:** |  |  |  |  |
|  |  |  |  |  |

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

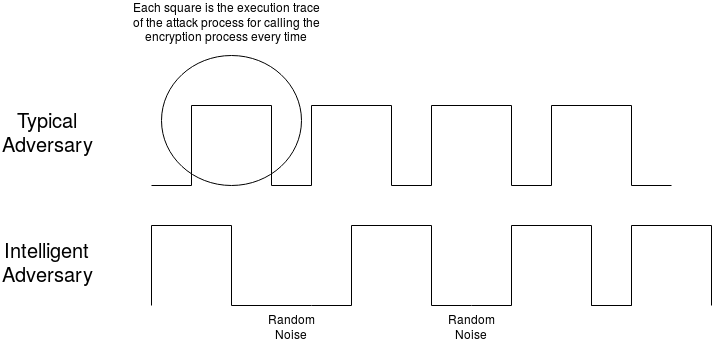
Many modern cryptosystems are vulnerable to side-channel attacks even after the recent advancement of cryptographic algorithms. These attacks compromise the secret key by the information gained through the physical implementation of the cryptosystem. One category of such attack includes micro-architectural 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 extra overhead. Implementations of these countermeasures are not feasible in resource constraint environment like IoT devices, Smartphones, etc. There have been some recent works [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. Most of these techniques articulate 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, which 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.

* 1. **Disadvantages of the recent works**

Recent state-of-the-art detection techniques as mentioned in [5, 6] detect the cache-based attacks with high efficiency, but the difference between a benign program with high micro-architectural activity and a side-channel attack cannot be confirmed with high confidence. These techniques do not deal with false positives efficiently. Moreover, there exist some “intelligent” adversaries which perform some random operations in between their workflow to confuse the detection mechanism. Even these works reported the inability to detect these attacks with their detection scheme, i.e., there is a chance of false negatives considering these types of attacks.

* 1. **Research Plan and Description**

A typical side-channel adversary needs to sequentially access the encryption algorithm multiple times to successfully retrieve the secret key. Current state-of-the-art techniques can detect these adversaries with high confidence. The execution of an “intelligent” adversary consists of attack part added with some random noise to fool the detection method. But, the limitation of these adversaries is that they 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 before the addition of the random noise. 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 because of the added noise. 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. 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 side-channel attack processes, i.e., cache-based attack, branch-based attack, etc., have these types of ‘special’ portions in their execution. If we can capture these parts of their executions, then we will be able to detect these types of attacks without any false negatives. Similarly, a benign process without having these 'special' portions in its execution can never be detected as a malicious process, thereby, reducing the number of false positives.

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, which is shown in the above example. These temporal patterns could be easily learned using Long Short-Term Memory (LSTM) Recurrent Neural Networks [8, 9]. The motivation of using LSTM Networks is that - LSTMs are special kinds of RNNs capable of learning long-term dependencies in a sequence. The advantage of LSTMs is that they are very generic and does not require any feature engineering beforehand, i.e., we do not need to select any important feature (i.e., which hardware event to monitor) for the decision. For a successful attack, the current execution of an adversary process needs to get information from the previous iterations. The addition of arbitrary noisy execution does not change this scenario, which signifies that the hardware trace of an adversary is not just a random trace but a dependent time-series data. To capture this dependency, we believe LSTMs are the best model in this scenario.

* 1. **Demonstration and experimentation plan**

The proposed experimentation to carry out this research is discussed below -

1. The sources of "intelligent" attacks are insufficient. To develop our detection methodology and to learn the LSTMs we need a significant variation of these attack examples. So, our first objective is to design and use such "intelligent" attacks along with the currently available sources. The development of these attacks depends on two forms of noise - one coming from other system programs, and another coming from the 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 sleep, 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. We started to produce this dataset by considering conventional cache timing attack on AES and the results obtained are shown below.

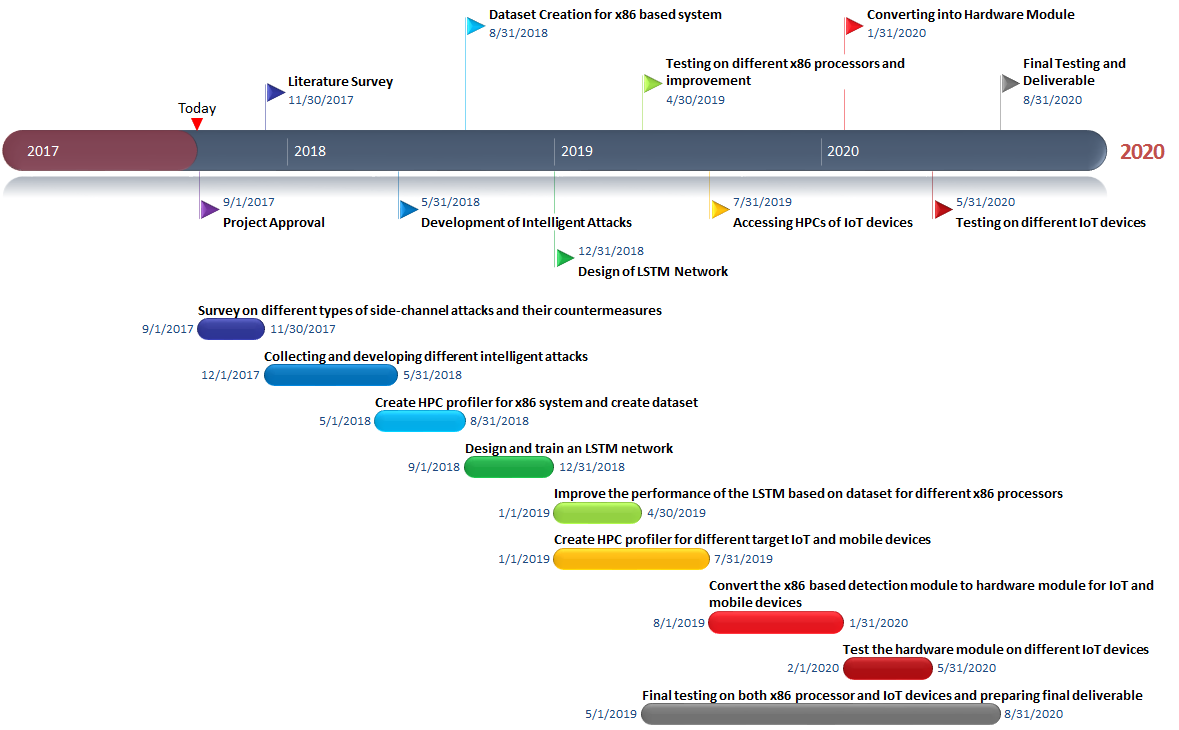
|  |  |  |
| --- | --- | --- |
|  |  |  |
| Cache Miss trace of AES encryption for a particular secret key. | Cache Miss trace of conventional cache-timing attack on AES | Cache Miss trace of modified intelligent attack on AES |
| Fig. 2: Temporal Sequences of different scenarios in the absence of background noise. | | |

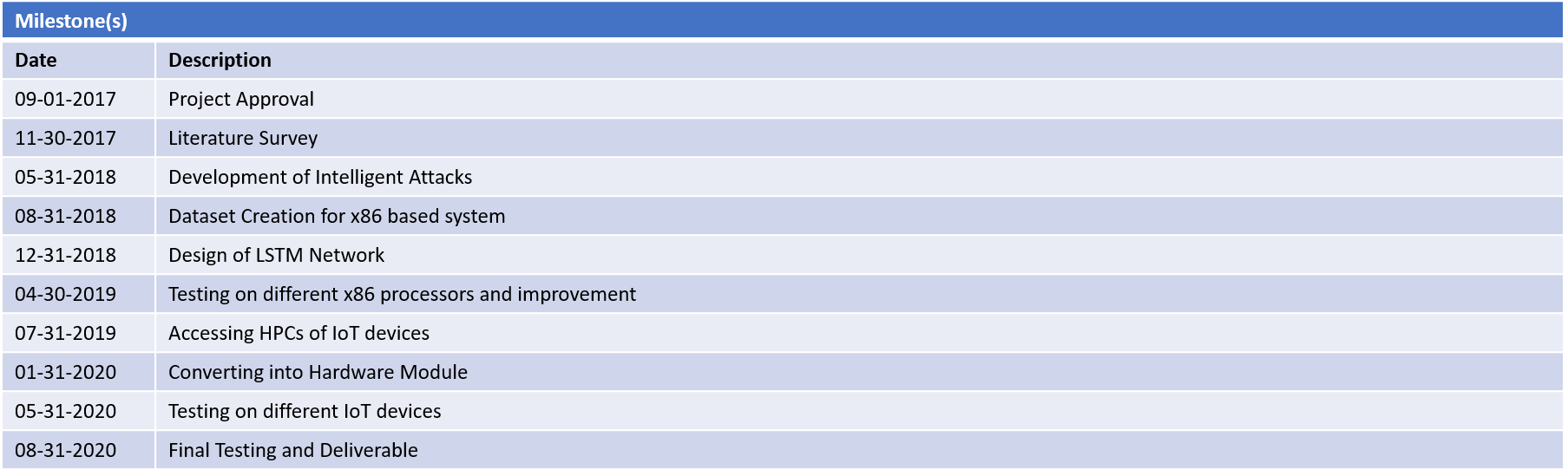
|  |  |  |
| --- | --- | --- |
|  |  |  |
| Cache Miss trace of AES encryption for a particular secret key. | Cache Miss trace of conventional cache-timing attack on AES | Cache Miss trace of modified intelligent attack on AES |
| Fig. 3: Temporal Sequences of different scenarios in the presence of background noise. | | |

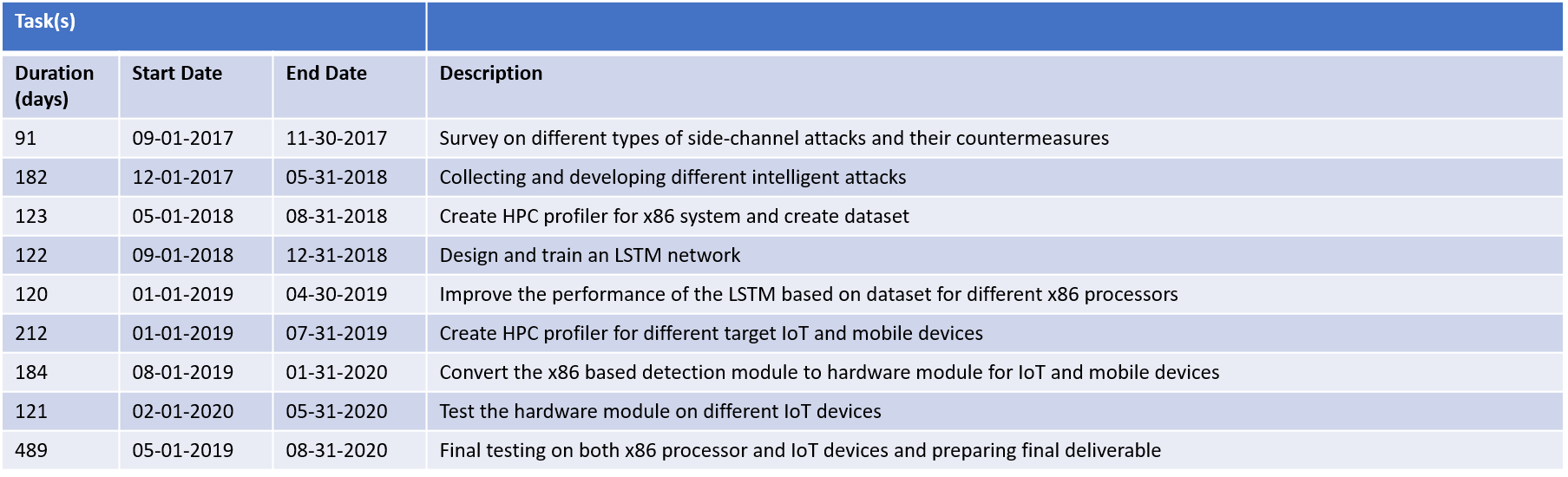
We can easily see from Fig. 2 and Fig. 3, that a side-channel adversary needs to access the encryption algorithm multiple times and we can see the presence of the trace of encryption algorithm within the trace of the adversary. Also, the intelligent adversary incorporates noise within its code segment, which we can also see form the above figures.

Our objective is to generate a dataset for most of the well-known encryption algorithms and corresponding side-channel attacks. Since we generate the dataset by ourselves, we know which portion of the code is actual attack code and which are random noise. This information will help the LSTM network to learn the behavior of the intelligent attack process accurately.

1. Once we obtain the dataset for our problem, our next task is to classify all the 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.
2. We propose to perform all the experiments on Intel and AMD x86 processors initially for the initial results and to tune the success rate of the propose solution, and then we will convert the detection module into hardware level with minimum implementation overhead to provide security to IoT and mobile devices.
3. **Organizational Remarks**
   1. **Statement of work, schedule, milestone and deliverables**







**Note:** The date of Milestones and Tasks varies according to the Project Approval date but the duration remains the same.

The final deliverables to Intel through this project are –

1. A secure software module for protection against side-channel attacks on x86 based platforms, designed using LSTM network.
2. A secure hardware module, developed using LSTM network, to provide security against side-channel attacks to IoT and mobile devices.
   1. **Proposal team**

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

* 1. **Cost volume**

Cost summary that documents the expected resources, expenses, overhead, and equipment in USD.

**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. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8) (1997) 1735–1780
9. Olah, C.: Understanding lstm networks (Aug 2015)