Hardware Security in India: The Journey so Far

8

Debdeep Mukhopadhyay

Abstract

Hardware Security is relatively a young discourse when compared to its more classical counterparts, like cryptography or network security. Yet, the growth of this subject in spite of his short history is phenomenal and the impetus of it in the modern-day world is striking. Like other parts of the world, India also joined this important area of research from the last decade to cover important areas in this discourse. This article is a description of some of the core subareas from our country, which includes cryptographic hardware design, side-channel analysis, fault analysis, micro-architectural attacks, Trojan detections, physically unclonable functions, and applications in IoT security. The chapter concludes with some ongoing efforts in the country to extend the core competence developed to develop secured end–end cyber-physical systems, which is of immense importance not only in our country, but also in the rest of the world.

Keywords

VLSI of cryptographic algorithms • Side-channel analysis • Countermeasures • Fault tolerance • Micro-architectural attacks • PUFs • IoT security

8.1 Introduction

Cryptology is the art of making and analyzing algorithms to provide confidentiality, integrity, and availability of information. This encompasses, design of ciphers, and analyzing them via the process of cryptanalysis; developing classical primitives like hash functions, message authentication codes, signature schemes, and the like. Recent days there has been a humongous advance in the field of cryptology, with advanced primitives like attribute-based encryption, functional encryption, fully homomorphic encryption, and last but not the least, post-quantum cryptosystems are being developed and analyzed. These primitives provide additional capabilities, like performing operations on encrypted databases in the pervasive public cloud, yet addressing privacy and security issues. While cryptology deals with the theoretical constructions, practitioners have shown that mathematically strong algorithms are just the beginning. Cryptographic algorithms, like Advanced Encryption Standard (AES), or RSA, Elliptic Curve Cryptography (ECC) when implemented on software or hardware in a naïve manner, there are potential information leakage sources, through what are called as side channels. In the pioneering works of Paul Kocher around 1995, it was reinstated that side-channel sources like timing or power can be utilized effectively to determine the complete secret key using efficient attack algorithms [1]. These side-channel analyses opened up new research directions in cryptographic engineering, which attempts at developing a theory of the reality, which is otherwise not modeled by classical cryptography. In 1997, with a seminal paper from Dan Boneh [2] and followed with works from Biham–Shamir [3], another class of popular attacks, called fault attacks, came to the surface. These attacks showed that popular cryptosystems like RSA or DES can leak the secret keys when there are accidental or unintentional faults in the circuit that computes them. These works show that proper cryptographic engineering does not end with only performance as a criteria, what we also need is protection against side channels and other implementation-driven attacks. At IIT Kharagpur, in order to explore this important and exciting world of cryptographic engineering, the Secured Embedded Architecture Laboratory (SEAL) was set up in 2008. This lab has made fundamental contributions to the space of research in cryptographic engineering, like developing the most efficient fault attack on the worldwide standard AES (Advanced Encryption Standard) cipher, designing fast and small ECC (Elliptic Curve Cryptosystems [4]) hardware IPs, etc. Furthermore, the laboratory has developed the infrastructure and practice to validate the claims through

D. Mukhopadhyay (✉)
Indian Institute of Technology Kharagpur, Kharagpur, India
e-mail: debdeep@cse.iitkgp.ac.in

S. K. Shukla and M. Agrawal (eds.), *Cyber Security in India*, IITK Directions 4,
https://doi.org/10.1007/978-981-15-1675-7_8

rigorous experiments, done on state-of-the-art platform for side-channel analysis.

Hardware security is a field which originates from cryptographic engineering, and applies it to develop a secured hardware layer in computing systems which can serve as a more dependable root-of-trust (RoT) than their software counterparts. Indeed earlier efforts to design the Trusted Platform Modules (TPM) chips, which offer a very low-cost cryptocoprocessor design, typically includes RSA key-generator, SHA-1 hash functions, and encryption–decryption signature schemes. The signature was used for signing and attesting codes, which are considered legitimate to execute. Computers that incorporate a TPM can create cryptographic keys and encrypt them so that they can only be decrypted by the TPM. This process, often called wrapping or binding a key, can help protect the key from disclosure. Each TPM has a master wrapping key, called the storage root key, which is stored within the TPM itself. The private portion of a storage root key or endorsement key that is created in a TPM is never exposed to any other component, software, process, or user. Cryptosystems that store encryption keys directly in the TPM without blinding could be at particular risk adversaries. In 2010, Christopher Tarnovsky presented an attack against TPMs at Black Hat, where he claimed to be able to extract secrets from a single TPM by inserting a probe and spying on an internal bus for the Infineon SLE 66 CL PC [5]. In 2015, reports on differential power analysis attack against TPMs potentially able to extract secrets, were reported [6]. Thus striving to find a hardware security primitive for usage as an RoT is of utmost importance. One of the accompanying challenges is to ensure that the design should be low cost. One promising technology is what are called as Physically Unclonable Functions (PUFs) [7]. The objective of PUFs is to create security primitives which provide every device with a unique fingerprint. PUFs in the black-box sense, receive inputs, which are called as challenges, to result in output response bits. Though there are different technologies on which PUFs can be designed, the most popular ones are silicon based. When the same PUF circuit is implemented on two different devices, the responses obtained for the same challenge to both of them are statistically independent. Thus PUFs offer novel designs of RoTs where the secret key is not explicitly stored in a memory and hence does not need to travel outside the IC boundary. However, PUFs are threatened by powerful machine learning attacks which try to mathematically model these primitives. One of the primary focus of SEAL, IIT KGP is to realize these primitives on hardware platforms, like FPGAs, ASICs, and perform rigorous test on them: with respect to machine learning attacks, reliability analysis with respect to temperature variations, humidity variations, etc.

PUFs are a promising security primitive to address the security concerns in Internet of Things (IoT). IoT products which are commercially being sold and deployed, like surveillance cameras, home automation systems, often have glaring security weaknesses. One of the goals of SEAL is to perform (in)security analysis of these commercially deployed devices. One of the primary reasons why ad hoc authentication mechanisms are often adopted in these devices are high cost of integrating standard cryptographic solutions. The adopted security solutions thus when adopted in the resource-constrained nodes need to minimize processing power, storage space, network bandwidth, energy consumption, and also perform with minimal user intervention. One of the major research goals that we strive is to integrate PUFs into the nodes. However, this also implies designing a new generation of security protocols which are based on the unclonability of PUFs. One of the major challenges for building such protocols is the fact that as PUFs are not clonable, the Challenge–Response Pair (CRP) which the verifier needs is to be explicitly stored. This makes the solution problematic as to store a large database in a secured fashion is cumbersome. One of the way-outs that we make an effort is to combine effectively PUFs with public-key cryptography to have a new generation of PUF-based authentication protocols which can be integrated with IoT subsystems. We aim to build surveillance cameras, smart meters, and other IoT products which have protections against various forms of adversaries like man-in-the-middle attacks, replay attacks due to the usage of PUFs.

IoTs or in general cyber-physical systems are heterogeneous. So, while on one end we have resource-constrained nodes and on other ends there are more powerful machines, servers, and finally the pervasive cloud. In these computing platforms, while side-channel analysis like power analysis, EM analysis may not be directly applicable, there is a very important variant of these attacks, which are called as micro-architectural attacks. These attacks target the underlying computer architecture, which has been fundamentally developed with performance in mind, and show that there are grave security flaws. Timing analysis, vulnerabilities due to the presence of cache memory, branch prediction attacks, row-hammer bugs of DRAM chips are some of the well-known micro-architectural attacks. At SEAL, we have been working on developing these attacks and exploring them on all well-known machines, like from Intel/ARM/AMD and have developed the expertise of not only demonstrating these attacks, but also developed coding practices for being secured against these attacks. These countermeasures are often costly and thus needs to be suitably designed to ensure a minimal footprint on performance. Another related research direction, which we pursue is to develop reactive measures by developing smart monitors in computing systems which can raise early alarms in the system on the presence of side-channel analysis. In this context, we show the use of machine learning based analysis for the detection of various micro-architectural side-channel attacks, like cache misses, branch-misses and even the newer Spectre and Meltdown.

Finally, no discussion on security is complete without bringing in the aspect of cloud. Clouds provide a massively powerful distributed computing paradigm, where service can be rendered and after usage can be relinquished. While on one hand cloud provides many opportunities for high-performance computing and is often conceptualized as the endpoint of the IoT, which can store and process billions of data which is being generated by the sensors, devices in IoT, they also raise grave concerns on security and privacy. Solutions to encrypt them, naturally raise questions on how to operate on encrypted databases. While theoretically feasible solutions like Fully Homomorphic Encryption (FHE) exist, which can serve as the holy grail in giving capabilities of performing arbitrary computations on encrypted data, but they are not feasible. As of now, they do not scale well and make operations like search extremely slow, virtually to the point of being unusable. At SEAL, we focus to develop on one hand new solutions for restricted but useful database computations, like search, using lightweight primitives, like pseudorandom permutations; on the other hand, we strive to leverage the parallelism offered by hardware designs to develop fast accelerators for such search algorithms.

In this article, we give an overview of the above topics, which are some of the highlights of contribution of hardware security research from our country. In particular, we provide the following descriptions which are some of the major hallmarks in hardware security research in India:

- Differential Fault Analysis (DFA)
- Hardware Design of Public-Key Cryptosystems
- PUFs: Design and Usage in IoT Security
- Microarchitectural Threats: Attacks, Countermeasures, and Detections
- Hardware Security to Accelerate Cloud Cryptosystems.

Organization: Sect. 8.2 of the chapter presents an overview of our work on differential fault attacks of cryptosystems, along with research in automating fault analysis. Chapter 3 discusses on VLSI design of public-key algorithms, with ECC in perspective. The chapter gives design approaches developed to cater to both design alternatives, viz. high-performance and lightweight applications. Subsequently, Chap. 4 focuses on PUF designs, briefing on works done on a promising PUF topology which combines machine learning robustness with reliability. It also presents a protocol suited for IoT applications using PUF-based primitives. Chapter 5 presents research efforts in the exciting confluence of computer architecture and security, annotating our findings in the topic of cache attacks, branch prediction attacks, and row-hammer attacks. We also discuss machine learning based detection methodologies for such side-channel attacks as an alternative for costly countermeasures. Finally, Chap. 6 presents a new line of research initiated in our laboratory to develop hardware-based accelerators for novel search algorithms on encrypted data. The overall direction of our research is summarized in Sect. 8.7.

8.2 Fault Analysis of Cryptosystems

Fault attacks are one of the most potent threats to modern secured systems. Given the fact that precise and targeted faults can be induced in most of the modern computing systems by means of commercially available low-cost equipment, fault attacks become reasonably practical. The SEAL Lab at IIT Kharagpur possesses several setups for targeted fault injection using intentional glitches in the clock network, electromagnetic radiation, and laser rays. The research team has successfully demonstrated injections for software and hardware implementations of crypto-algorithms. Practical realization of remote fault injections using the row-hammer vulnerability has also been demonstrated by the team. Evaluation of cryptographic primitives against fault attacks is an active area of research. The general philosophy for key extraction is analyzing both the correct and the faulty outputs from the target primitive. However, the analyses are mathematically intensive and widely vary among different primitives. This article will focus on fault attacks on block ciphers which are, undoubtedly, the most widely deployed cryptographic primitive so far. More specifically, we shall focus on a specific class of attacks called Differential Fault Analysis (DFA), which is the most widely explored and general class of fault analysis.[1] The fault analyses vary depending on the underlying structures of the block ciphers. However, the minimal requirement for an attack is the availability of at least one faulty ciphertext and its corresponding correct ciphertext.

8.2.1 Attacks and Countermeasures

Most of the early efforts of block cipher fault analysis were targeted toward the AES algorithm, which is the current worldwide standard for symmetric key encryption. AES is a Substitution-Permutation-Network (SPN) structure having 10, 12, or 14 iterative rounds, three different key sizes of 128, 192, and 256 bits, and three different state sizes of same lengths with the key. Each of the rounds consists of four bijective Boolean functions defined over $GF(2^8)$, among which one is nonlinear (`SubBytes`), and rest are linear (`ShiftRows`, `MixColumns`, and `AddRoundKey`). One of the major challenges at the initial phase of fault attack research was to evaluate how easily and practically one can extract the complete secret key of AES. One of the seminal works

[1] Other classes of fault attacks use certain target-specific physical assumptions over the nature of the faults. Their analyses are different but heavily influenced by DFA.

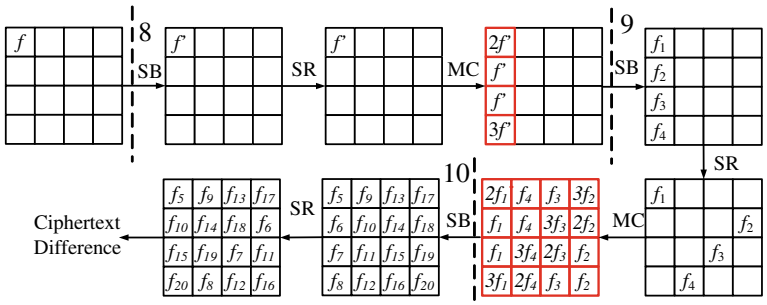


Fig. 8.1 Fault propagation in AES with the fault injected at the beginning of 8th round. The intermediate states which are utilized for constructing equations are marked in red

in this context was due to Tunstall and Mukhopadhyay [8], which showed that one single fault corrupting a byte of the intermediate state of AES at the beginning of 8th round, can reduce the size of the keyspace from 2^{128} to 2^8. As a result, the secret key can be compromised within minutes, even with nominal computational power. Furthermore, this attack has been proven to be optimal from information-theoretical perspectives. Several practical implementations of this specific attack, both on hardware and software AES implementations have been presented in the literature.

To provide a basic understanding of the fault analyses, here we present a brief description of the aforementioned attack. Let us assume a byte fault has been injected at the intermediate state of AES at the beginning of the 8th round. The value of the fault is unknown to the attacker, who can only observe the correct and the faulty ciphertext and guess keys. Each intermediate state, by convention, is represented as a 4×4 matrix of 16 bytes. Without loss of generality, we assume the 0th byte in the matrix to be corrupted. The XOR differential of the correct and faulty states are considered for analysis. As shown in Fig. 8.1, the fault propagates to the ciphertext, eventually corrupting the complete state of the cipher. Here, we have shown the differential propagation of the fault for convenience.

Referring to the differential propagation of the fault in Fig. 8.1, it can be observed that the state at the input of 10th round has certain observable linear patterns in its columns. More specifically, each of the columns is linearly dependent and spanned by a single variable. Exploiting these patterns four independent nonlinear system of equations (over GF(2^8)) can be constructed over associated keys, ciphertext differentials, and fault variables. One of these systems is shown in Eq. (8.1). Here each C_i denotes a byte from the ciphertext, and each k_i denotes the associated key byte. The faults are represented with the variable f_1.

$$\begin{aligned} 2f_1 &= S^{-1}(C_1 \oplus k_1) \oplus S^{-1}(C_1 \oplus \delta_1 \oplus k_1) \\ f_1 &= S^{-1}(C_{14} \oplus k_{14}) \oplus S^{-1}(C_{14} \oplus \delta_{14} \oplus k_{14}) \\ f_1 &= S^{-1}(C_{11} \oplus k_{11}) \oplus S^{-1}(C_{11} \oplus \delta_{11} \oplus k_{11}) \\ 3f_1 &= S^{-1}(C_9 \oplus k_9) \oplus S^{-1}(C_9 \oplus \delta_9 \oplus k_9) \end{aligned} \quad (8.1)$$

Solving this system for the 4 associated key bytes is supposed to filter out total 2^8 key choices. Solving all four such systems will reduce the entire keyspace from 2^{128} to 2^{32}. Further, a similar equation system can be constructed for the first column of the 9th round input state differential. This equation system will finally reduce the keyspace to a size of 2^8. The attack will remain exactly the same if any other byte of 8th round input gets corrupted. The attack still works even if the exact location of the corrupted byte at 8th round is unknown to the attacker. The keyspace size, in that case, becomes 2^{12}.

Fault attacks are inevitable for most of the cryptoprimitives present today. However, there is a consistent effort for building effective countermeasures against fault attacks. A large class of fault attack countermeasures performs some redundant computation to detect the presence of an injected fault. Redundancy can be realized via multiple encryptions over the same data at the simplest case, or via certain error correcting codes. However, it is not impossible for a clever attacker to bypass such redundant computations either by injecting the same faults in all computation branches or by finding gaps in the error detection capabilities of the codes used. One practical remedy to such advanced attacks is to make the computations randomized by cleverly introducing dummy rounds so that it becomes difficult for the attacker to identify the exact injection location. Moreover, explicit checks between the actual and redundant computations can be avoided. These facts give rise to an entirely different class of countermeasures called infective countermeasures. Infective countermeasures avoid explicit checks and randomize the output upon detection of a fault. To further enhance the randomization it performs dummy round computations between actual and redundant cipher rounds. An efficient infective countermeasure algorithm has been developed by Tupsamudre, Bisht, and Mukhopadhyay in [9].

8.2.2 Automated Detection of Fault Attacks

As already pointed out at the beginning of this section, the attack techniques vary significantly among different ciphers. Even for one block cipher, the entire analysis will vary depending on the location and nature of the fault. Due to such

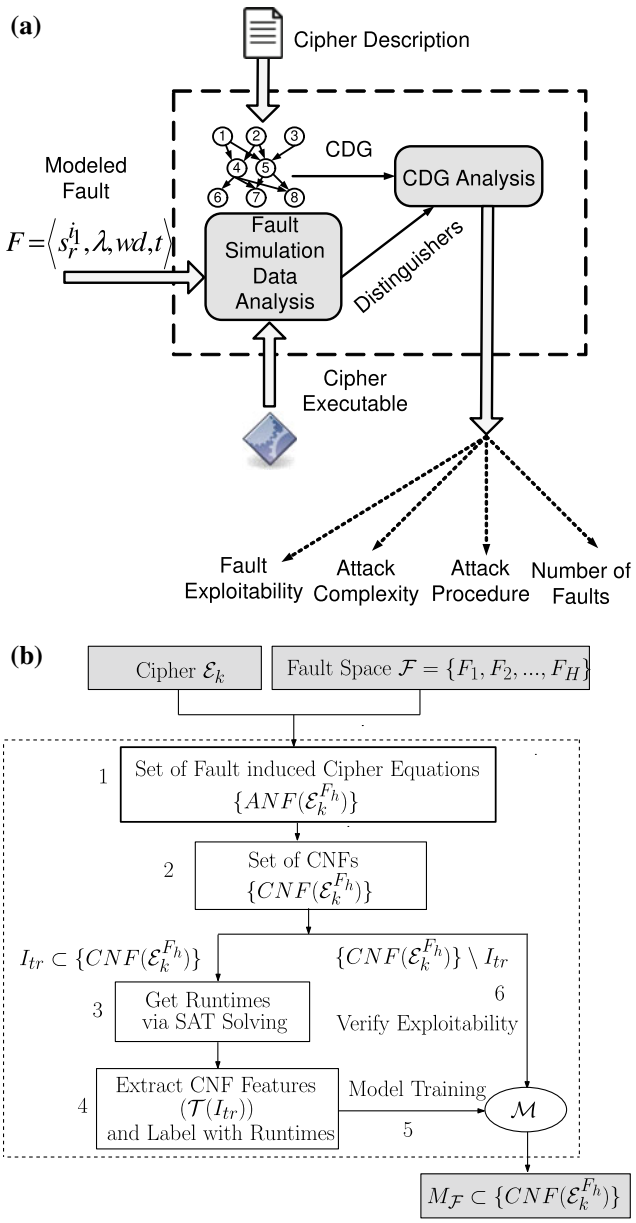


Fig. 8.2 Two automations **a** ExpFault framework [10]; **b** ML-Fault framework [11]

variability, fault attacks are nontrivial to generalize. However, it is not impossible as we shall show in this subsection. The availability of hundreds of block cipher designs makes the generalization and automation of fault attacks essential. Further, the fault spaces for each of these ciphers are prohibitively large in size. Any automation for fault attacks must be fast enough to cover the entire (or at least a significant share) fault space of a cipher.

The three expected properties of a fault attack automation are genericness, speed, and interpretability. An automation has been proposed in [10], which satisfies all these criteria simultaneously. Instead of performing the attacks explicitly, the automation proposed in [10] returns the attack algorithm and its complexity given the specification of a block cipher and a fault model. The necessary criteria behind every fault attack is a wrong key distinguisher, which can filter out a large part of the keyspace thus reducing the complexity of an exhaustive search. As a concrete example of a distinguisher, we refer to the 10th round input differential state in Fig. 8.1. The concept of distinguishers has been formalized in [10] based on Shannon Entropy. Based on the formalization, a fault simulation-based framework has been proposed, which mines out distinguishers from fault simulation data. The next step is to use a graph-based abstraction of the cipher, known as Cipher Dependency Graph (CDG) to figure out the exact attack algorithm (if any) and its complexity. A schematic of this framework, which is called as *ExpFault* is presented in Fig. 8.2a. ExpFault successfully figured out all previously proposed attacks on AES, and PRESENT block ciphers within minutes. Moreover, it figured out optimal attacks on a completely new cipher design called GIFT. Table 8.1 presents a summary of the attacks figured out for the GIFT cipher.

Although ExpFault is able to figure out the fault attacks successfully, it utilizes certain abstractions for the sake of scalability. For example, the internal structures of the S-Boxes are not considered explicitly in ExpFault. Also, it does not explicitly consider the fault values and plaintext values while sketching the attack path. Although these two parameters are not necessary to construct the attack paths in DFA, for certain cipher designs they may influence the attack complexities to some extent. As a result of these approximations, ExpFault returns the best possible attack complexity from the perspective of an attacker. While this information is often sufficient from a cipher evaluator's perspective, it cannot completely justify how an attack may perform on average on a given cipher design.

The only way to handle the above-mentioned issue is to get rid of all abstractions made in ExpFault. It is, however, feasible by means of an algebraic representation of the cipher. Algebraic representation converts each constituent Boolean function of the cipher (and the faults) to a system of nonlinear polynomial equations over $GF(2)$. This system then can be converted to Conjunctive Normal Form (CNF) and solved by Boolean Satisfiability (SAT) solvers. The attack is considered successful if the SAT solver returns the key within a reasonable time. The solver runs forever if the fault is not exploitable. Although SAT solving based approach is fairly generic, it is not fast enough for evaluating each fault. Further, the attacks are not interpretable. To make certain quantitative decisions on the fault space of a cipher, one thus require to make this approach scalable. To improve the scalability, a machine learning (ML) assisted framework is proposed in [11]. The ML engine is able to determine whether a SAT instance is solvable within a reasonable time just from the structure of the CNF representation. As a result, the fault

Table 8.1 Summary of DFA attacks on GIFT. We consider a fault injection attempt a successful attack only if both the evaluation complexity and remaining keyspace size is less than the size of the actual keyspace

Fault width	Round	Attack results				
		Evaluation complexity	$\lvert\mathcal{R}\rvert$	No. faults per location	Keys extracted	Comments
4	24	—	—	—	—	No attack found
	25, 23	$2^{17.53}$	$2^{7.06}$	1	128	Best attack found
	26, 24	2^{6}	$2^{3.53}$	1	104	Cannot extract full key
	27, 25	2^{6}	$2^{3.53}$	1	72	Cannot extract full key
8	24	—	—	—	—	No attack found
	25, 23	$2^{17.53}$	$2^{7.06}$	1	128	Best attack found
	26, 24	2^{6}	$2^{3.53}$	1	104	Cannot extract full key
	27, 25	2^{6}	$2^{3.53}$	1	72	Cannot extract full key

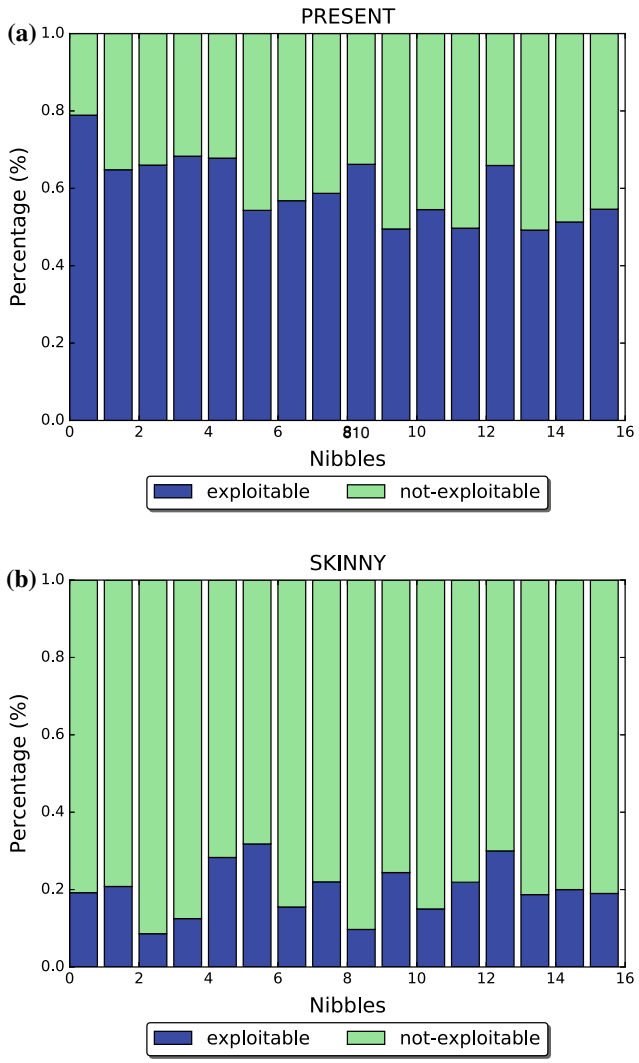


Fig. 8.3 Exploitable fault spaces with **a** PRESENT S-Box; and **b** SKINNY S-Box [11]

characterization becomes extremely fast after a certain number of fault instances are exhaustively characterized with SAT for training. This brings the ability to characterize a large sample of faults within a reasonable time. A schematic of this framework is given in Fig. 8.2b. Several interesting observations can be from these large characterized fault samples.

To illustrate the importance of this fault characterization framework, we present an example adapted from [11]. In this example, the PRESENT block cipher structure is instantiated with two different S-Boxes having similar mathematical properties. Figure 8.3 presents the characterized fault space for these two instantiations corresponding to a specific fault location. It can be observed that the percentage of successful attack instances are significantly different in these two cases, depending on the value of the injected faults. This observation leads to the discovery of certain mathematical properties of the S-Boxes, which were not previously known to have any effect on fault attacks (see [11] for further details). From another perspective, these two graphs represent the average success rate of an attacker for a given fault model, which cannot be trivially obtained from ExpFault at its current stage. However, this ML-based framework cannot provide exact complexity figures and, more importantly, the attack paths. So both the frameworks are somewhat complementary.

8.3 Hardware Design of Public-Key Cryptosystems

Public-key cryptography plays a very important part in a secure communication network. It ensures confidentiality, integrity, and authenticity of the information being exchanged and performs the secure exchange of secret keys for execution of symmetric-key ciphers. Among the available public-key

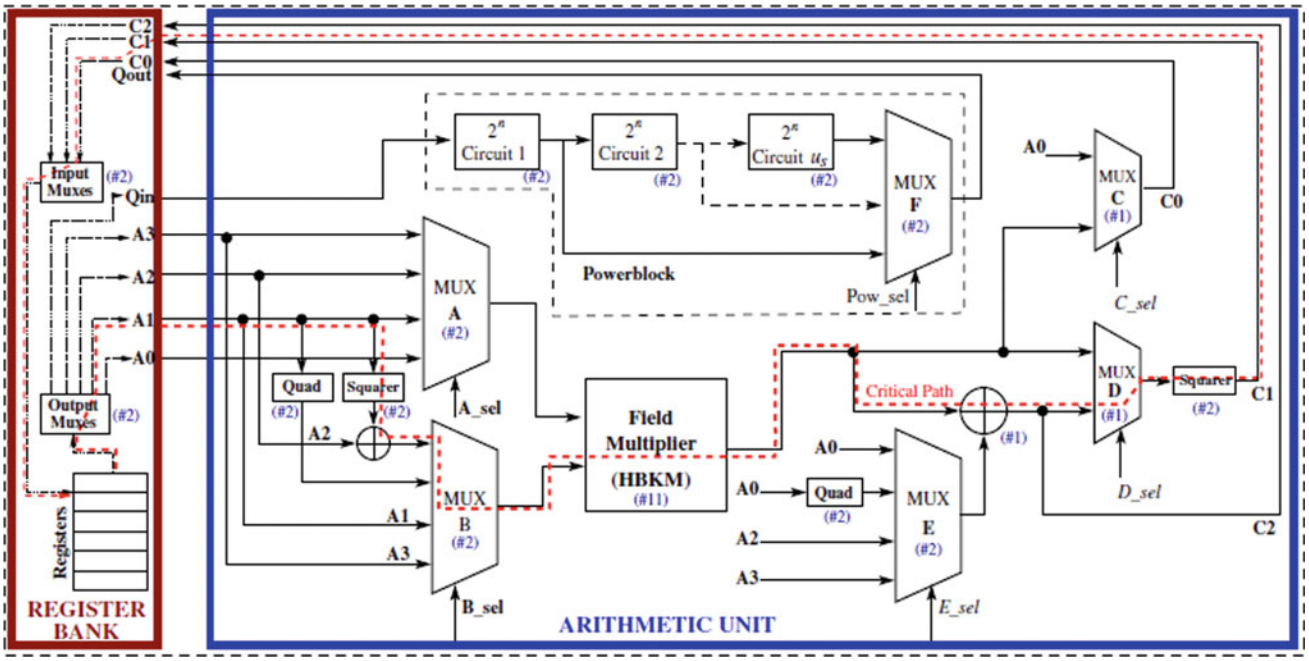


Fig. 8.4 Block diagram of the proposed ECC scalar multiplier architecture in $GF(2^n)$ [13]

cryptosystems, RSA and Elliptic Curve Cryptography (ECC) are the most prominent. The mathematical foundation of ECC is based on the intractability of discrete logarithm problem in elliptic curves. Compared to RSA, ECC provides more security per key bit. However, the computation steps involved in the execution of ECC are mathematically intensive, making its software implementation inefficient and unsuitable for many real-world applications. An alternative approach is to provide dedicated crypto-accelerator on hardware platforms like ASIC (Application-Specific Integrated Circuit) and FPGAs (Field Programmable Gate Arrays) to accelerate ECC scalar multiplication [12]. The execution of ECC is generally carried out in either $GF(2^n)$ or in $GF(p)$. In the SEAL Lab, we have developed efficient implementations for ECC-based cryptography for both $GF(2^n)$ and $GF(p)$ for FPGA platforms which we have summarized in subsequent sections.

8.3.1 Fast and Efficient Implementation of $GF(2^n)$ ECC Scalar Multiplication on FPGA

The most important operation for execution of ECC is elliptic curve scalar multiplication. A typical execution of ECC scalar multiplication takes a point on the curve P and a scalar k as input and produces $[k]P$ as output. In this subsection, we will discuss our proposed high-speed implementation of ECC scalar multiplication in $GF(2^n)$ [13,14]. The architecture of the proposed implementation is shown in Fig. 8.4. The entire architecture for executing ECC scalar multiplication in $GF(2^{163})$ consumes around 148 registers, 10195 lookup

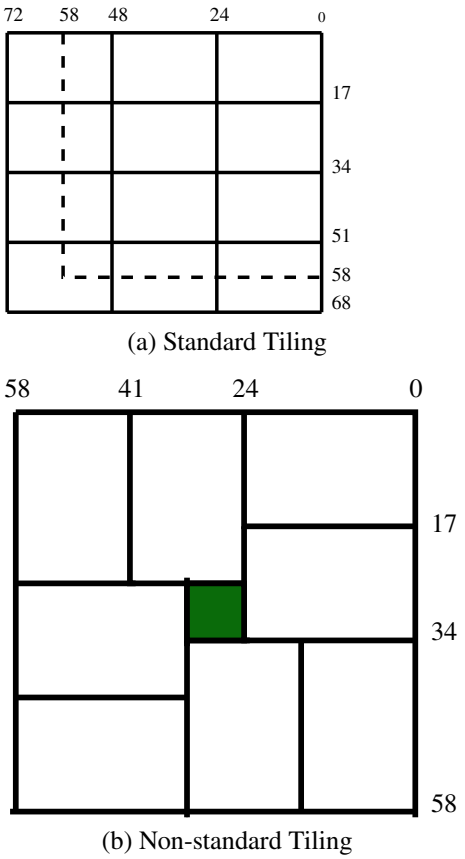


Fig. 8.5 Multiplying operands of width 58 using asymmetric multipliers [15]

tables (LUTs), and 3513 logic slices on a Virtex-5 FPGA. The corresponding latency of the design is only 9.5 µs. This performance escalation is achieved due to multiple architectural optimization techniques that we have incorporated in the design. Few of them are discussed below:

- **Increasing clock frequency with optimal pipelining:** The performance of any hardware implementation is highly dependent on the critical path of the architecture. Pipelining is a common approach which is incorporated in the design to reduce the critical path so that the design can support high frequency of operation. However, pipelining also increases clock cycle requirement of the design. It has been found that blindly applying pipelining often deteriorates the performance of a design as reduction of critical path delay is nullified and usually overshadowed by the increment in the clock cycle requirement. In this work, we have modeled the delay of different mathematical operations like field adders, field multipliers, exponentiation modules. These delay values allow us to partition the critical path in an optimal manner so that the advantage of the pipelining technique is truly assimilated in the architecture.
- **Reducing clock cycle requirement:** Another strategy to improve the performance of the design is to reduce the clock cycle requirement of the architecture. To achieve this, we have first developed efficient and fast implementation of field multiplier and exponentiation module in $GF(2^n)$. The field multiplier architecture is based on hybrid Karatsuba multiplication algorithm and the exponentiation module is implemented using Itoh–Tsujii algorithm. The architecture for Itoh–Tsujii algorithm is generally based on field squarer module. However, for FPGA platform, the performance of Itoh–Tsujii algorithm can be improved significantly if it is implemented using field quad module (computing fourth power of the input). Additionally, we have improved the scheduling of the scalar multiplication algorithm by overlapping the mathematical operations of two consecutive scalar bits.

We have validated the proposed architecture for ECC scalar multiplication in $GF(2^{233})$ also. Compared to the existing implementations, our proposed implementation was fastest without having any significant area overhead among reported literature at the time of publishing the work.

8.3.2 Efficient Resource Utilization for ECC Scalar Multiplication in $GF(p)$

In this section, we will focus on the ECC scalar multiplication in $GF(p)$. ECC implementation in $GF(p)$ on FPGA can be significantly improved by deploying FPGA-based hard-IPs like *DSP blocks* and *block RAMs*. In this context, we have proposed a generalized *non-standard tiling methodology* in [15]. The proposed methodology in [15] focuses on the optimum usage of DSP blocks with the objective of faster field multiplications. The multipliers present inside the DSP blocks of modern FPGAs are asymmetric in nature. They are capable of computing 24×17 unsigned multiplication. Due to this asymmetric multipliers, standard school book method of field multiplication will be non-optimum. Hence, we have introduced the nonstandard methodology for field multiplication, which makes the usage of DSP blocks optimum. An example of nonstandard tiling is shown in Fig. 8.5. In [15], we have generalized the notion of nonstandard tiling so that nonstandard tiling can be applied to multipliers with large operand width, which is the typical scenario in case of ECC. An illustration of our proposed methodology is shown in Fig. 8.6.

Using the proposed nonstandard tiling methodology, we have developed a fast field multiplier which uses the DSP blocks in an optimal manner. The application of nonstandard

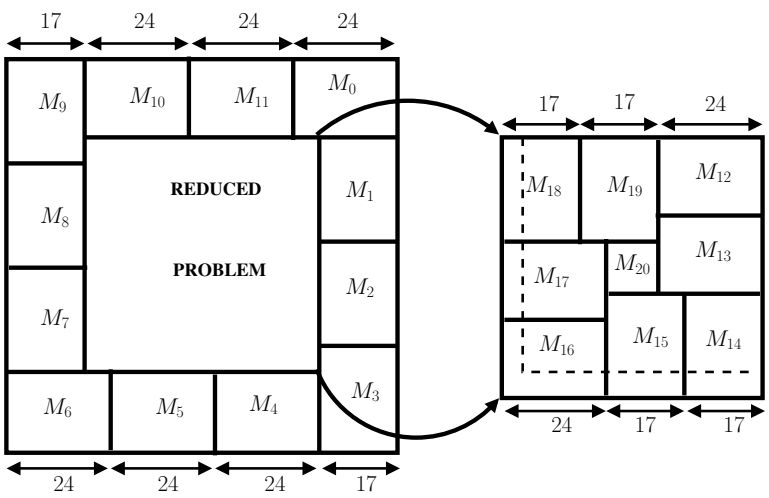


Fig. 8.6 Multiplying operands of width 89 using asymmetric multiplier of dimensions 24 and 17 [15]

Operand width(b)	Mapped Operand width(a)	Decomposition	Reduction Step	Multipliers Required by standard Tiling	Multiplier Required by Non-Standard Tiling
192	191	$24*3+17*7$	$191 \rightarrow 47$	96	90
224	222	$24*5+17*6$	$222 \rightarrow 18$	140	120
256	256	$24*5+17*8$	$256 \rightarrow 16$	176	160
384	382	$24*6+17*14$	$382 \rightarrow 94$	368	360
521	519	$24*11+17*15$	$519 \rightarrow 9$	682	660

Fig. 8.7 Nonstandard tiling vs. standard tiling [15]

Table 8.2 Different variant of SBN instruction

Instruction	Memory write-back	Multiplier reset	Key-shift	Right-shift
$SBN_{\overline{wmulksrs}}$	✓	x	x	x
$SBN_{nw\overline{mulksrs}}$	x	x	x	x
$SBN_{nwm\overline{ulksrs}}$	x	x	✓	x
$SBN_{\overline{wmulksrs}}$	✓	x	x	✓
$SBN_{nwmul\overline{ksrs}}$	x	✓	x	x

tiling for NIST curves specified for ECC scalar multiplication in $GF(p)$ and the corresponding DSP block requirement is shown in Fig. 8.7.

8.3.3 Lightweight Architecture for ECC Scalar Multiplication in $GF(p)$

In this work [16], we have proposed a single instruction approach for implementation of ECC scalar multiplier suitable for lightweight applications. More specifically, we have built an entire ECC scalar multiplier processor by using only one URISC (Ultimate Reduced Instruction Set Computer) instruction SBN (subtract and branch if negative). It is well known that using such URISC instruction, one can execute any logical or mathematical operation, leading to a Turing complete computer processor. However, as ECC involves many computationally intensive field operations, a standalone SBN-processor for ECC scalar multiplier execution will be drastically slow.

To tackle this problem, we have integrated the SBN processor with dedicated hard-IPs (Block RAM, DSP Blocks) of the modern FPGAs to demonstrate an implementation of immensely lightweight, yet practical ECC architecture [16]. We have proposed four different flags in the processor architectures which when set, activates different optimization strategies integrated inside the processor architecture. The optimized strategies proposed by us are: the option of switching off memory write back, dedicated right shifter, and dedicated field multiplier built exclusively with DSP blocks. It must be noted that field multiplication and right shift can also be executed by repeated execution of SBN instruction, but that will be extremely slow and inefficient. Hence, we have developed efficient architectures for these operations using FPGA-based hard-IPs. The details of the proposed SBN instruction along with four different flags are shown in Table 8.2. The architecture of the proposed SBN-processor for ECC scalar multiplication is shown in Fig. 8.8.

For comparing with other existing implementation, we have implemented *NIST P-256* ECC scalar multiplier using SBN processor on Virtex-5 and Spartan-6 platform. In both cases, the slice consumption of the design is less than 100. To the best of our knowledge, this is the first implementation of ECC which requires less than 100 slices on any FPGA device family. The stand-alone SBN processor is itself very lightweight, and the dedicated multiplier core is designed by judicious use of DSPs. Additionally, the block RAMs are used extensively to implement both data and instruction memory of SBN-ECC processor. It must be noted that a designer can choose a budget of slices and block RAMs and can design the corresponding SBN-ECC processor as per his choice. Moreover, the timing performance of the SBN-ECC processor is comparable with the existing implementations. This may seem to be counterintuitive as the proposed SBN processor needs to execute a large number of instructions to complete one single scalar multiplication. However, it must be noted that the proposed SBN processor is coupled with a dedicated field multiplier built using high-performance DSP blocks. This improves the timing performance of the proposed architecture significantly.

8.4 PUFs: Design and Usage in IoT Security

The idea behind PUFs is to utilize device-specific random intrinsic features for identification. Depending on the technology used for implementation, PUFs can be categorized into four categories: Optical PUF, Silicon PUF, Coating PUF,

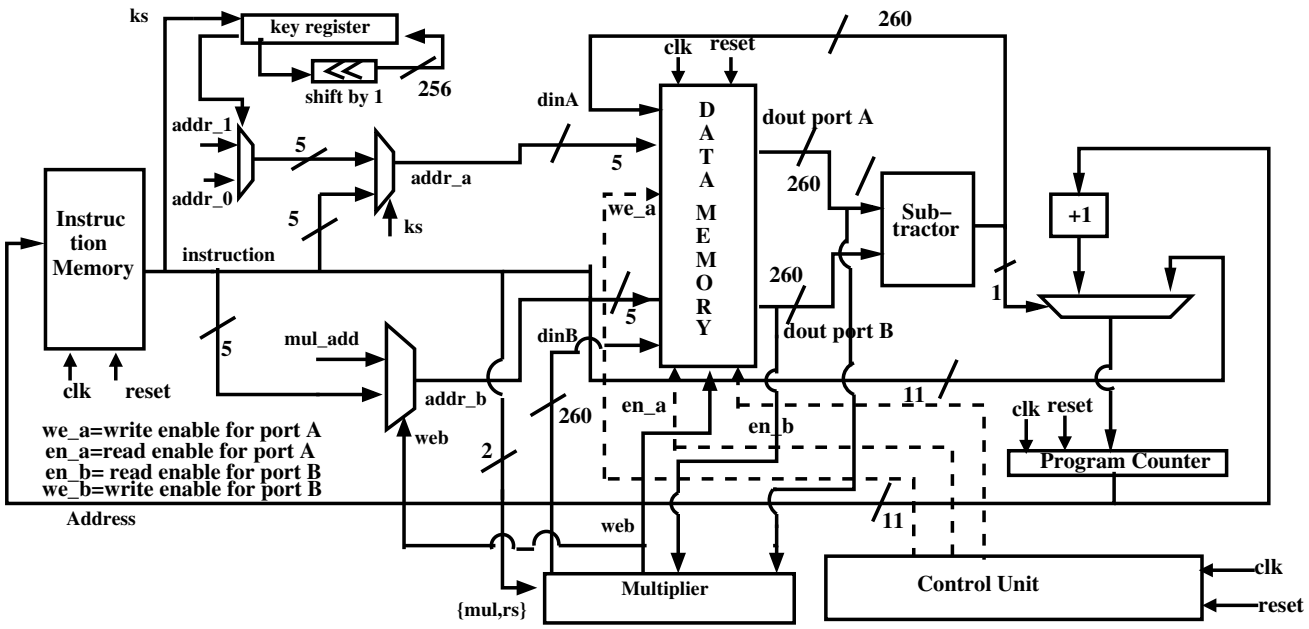


Fig. 8.8 ECC SBN processor architecture [16]

and Acoustic PUF, out of which Silicon PUFs have gained much attention. A silicon PUF is a one-way function embedded in an IC which utilizes the uncontrollable and intrinsic physical characteristics of the IC occurring due to the variations in the manufacturing process. The output of a PUF instance, or *response* is uniquely determined by the input, also known as *challenge* and device-specific variations. An applied challenge and its measured response is generally called a *Challenge–Response Pair* or CRP and the relation enforced between challenges and responses by one particular PUF is referred to as its CRP behavior. Every PUF instance exhibits a unique and unpredictable challenge–response behavior that is hard to characterize physically or mathematically, but is otherwise easily and reliably evaluated, which makes PUF a good candidate for various security applications such as random number generation, key generation, hardware authentication, etc.

Ever since the advent of PUFs, multiple attempts have been made to compromise the security of the hosting device or application by learning the embedded PUF behavior. Machine Learning attacks, being a noninvasive method, have been a very popular choice. In such attack scenarios, a relatively small subset of challenges along with their respective responses is collected by the adversary, attempting to come up with a model describing the challenge–response behavior of the PUF, and are commonly known as classical ML Attacks. An effective countermeasure for this attack is to increase the modeling complexity of the learning algorithm, which in turn would require more training data to create a fairly accurate model. It was later pointed out by Becker in [17] that beside the challenge– response behavior, reliability of a PUF also holds essential information. Reliability of a PUF is a measure to check similarity in response to a particular challenge when applied multiple times to a particular PUF under different environmental conditions. On investigating for Arbiter PUF (APUF), which is an intrinsic delay-based PUF, it is observed that the delay difference ΔD for a specific challenge, which decides the response of APUF is directly proportional to the unreliability of the corresponding response bit if the environmental conditions are kept stable. This is due to the fact that the various sources of noise add an approximately Gaussian delay $D_{noise} = norm(\mu, \sigma)$ to the delay difference ΔD. If the delay difference ΔD_{PUF} for a given challenge C is very large, it is unlikely that the noise term D_{noise} changes the sign of ΔD. Hence the response bit remains the same. However, if the delay difference ΔD_{PUF} is close to zero, then it is much likely to change the response bit. Change in the environmental conditions such as temperature or supply voltage has a similar or rather a stronger effect than thermal noise over the reliability of response. Based on this observation, Becker proposed a reliability-based machine learning attack in [17] using CMA-ES (Covariance Matrix Adaptation-Evolution Strategies) algorithm, which selects a set of models whose reliability vector fits closely to the actual reliability vector of the PUF to be modeled. In each round, certain modifications are made to the selected models and using the fitness criteria, the next generation of PUF models are selected.

Considering these factors, we at SEAL, IIT Kharagpur, have designed a MUX-based Arbiter PUF composition (MPUF) [18], denoted by (n, k)-MPUF, as shown in Fig. 8.9 with the objective of improving modeling robustness and reliability. As shown in Fig. 8.9, there are n data input APUFs and k selection input APUFs and the response of

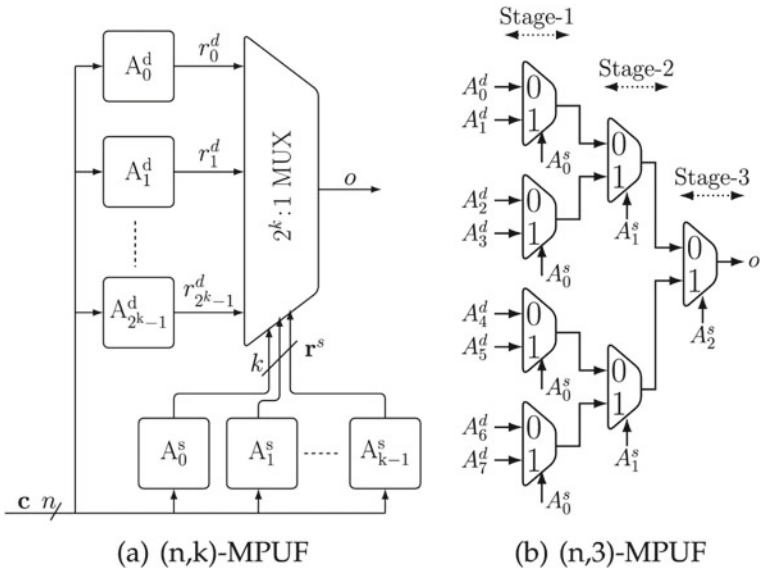


Fig. 8.9 Block diagram of a MUX PUF circuit [18]

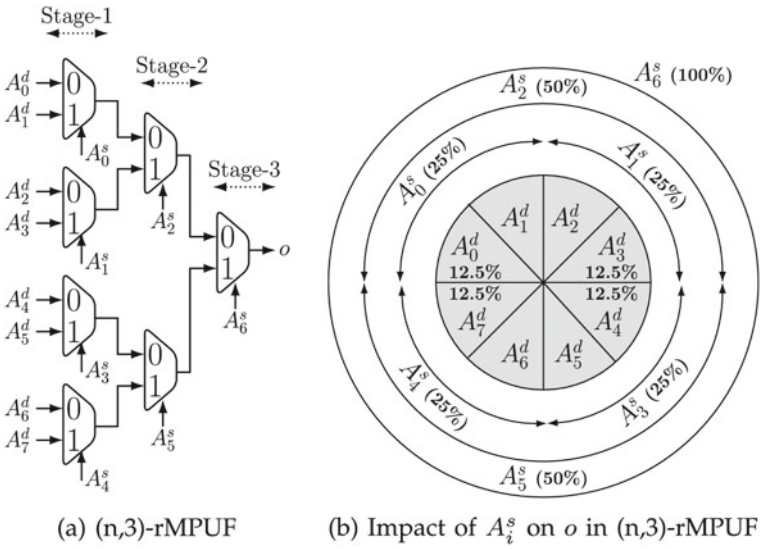


Fig. 8.10 Block diagram of a Robust MUX PUF (rMPUF) circuit [18]

(n, k)-MPUF depends on k selection input APUF and one input data APUF, which is selected by the output of selection APUFs. This implies that the final response is independent of remaining $(n - 1)$ APUFs, which results in better reliability. Along with this, a robust MPUF variant, shown in Fig. 8.10, is also proposed which aims to reduce the contribution of selector inputs APUFs on the MPUF CRP space, thereby reducing the correlation between reliability of MPUF and selection APUF outputs, thus making it robust against reliability-based modeling attack. Reduction of contribution in CRP space implies that an adversary would need more CRPs to build a high accuracy model which is required in the training phase of such attacks (Fig. 8.11).

For an (n, k)-rMPUF, the total number of distinct CRPs required for modeling all selection input APUFs is

$$N_k^S = N_k^S + \sum_{j=1}^{k-1} \left(\frac{N_k^S}{2} \times 2^j \right) = 2^{(k-1)} N_k^S \tag{8.2}$$

where N_k^S denotes number of CRPs required for reliability-based modeling of a selection input APUF instance in (n, k)-rMPUF. Thus modeling complexity increases exponentially with respect to number of selection input APUFs k which makes rMPUF robust against Becker's attack.

Moreover, MPUF and its variants have near ideal uniformity and uniqueness values for various values of n, k, and α, as shown in Table 8.3. This work is an attempt to explore the capabilities of MPUF and its variants across various PUF performance metrics and to provide a better alternative to XOR-PUF, which is considered a good choice in PUF-based

Table 8.3 Performance metrics (%) of simulated (n, k)-MPUF/cMPUF/rMPUF

PUF	n	k	$\alpha^\star$	Uniformity (Avg., Std.)	Uniqueness (Avg., Std.)	Reliability (Avg., Std.)			$x^\dagger$
						(n, k)-MPUF Variant	(k+1)-XOR APUF	x-XOR APUF	
MPUF	64	3	1/2	(50.82, 3.32)	(50.04, 1.43)	(86.24, 0.63)	(77.47, 1.22)	(53.60, 0.50)	11
			1/20	(50.33, 2.72)	(50.02, 0.61)	(98.91, 0.05)	(98.28, 0.10)	(94.62, 0.23)	
			1/80	(50.33, 2.71)	(50.02, 0.61)	(99.74, 0.01)	(99.59, 0.03)	(98.82, 0.05)	
		4	1/2	(49.57, 2.03)	(50.01, 0.61)	(83.57, 0.59)	(71.99, 1.07)	(50.16, 0.09)	20
			1/20	(49.80, 1.67)	(50.01, 0.32)	(98.67, 0.06)	(97.82, 0.11)	(89.40, 0.35)	
			1/80	(49.79, 1.66)	(50.01, 0.32)	(99.69, 0.01)	(99.49, 0.02)	(97.79, 0.06)	
	128	3	1/2	(49.68, 2.08)	(50.02, 0.64)	(86.40, 0.48)	(77.83, 0.91)	(53.74, 0.40)	11
			1/20	(49.79, 1.82)	(50.01, 0.39)	(98.92, 0.04)	(98.31, 0.08)	(94.69, 0.15)	
			1/80	(49.79, 1.81)	(50.01, 0.39)	(99.74, 0.01)	(99.60, 0.02)	(98.84, 0.04)	
		4	1/2	(49.87, 2.24)	(50.00, 0.39)	(83.61, 0.36)	(72.13, 0.74)	(50.15, 0.08)	20
			1/20	(49.95, 1.73)	(50.00, 0.21)	(98.68, 0.04)	(97.85, 0.08)	(89.47, 0.25)	
			1/80	(49.95, 1.72)	(50.00, 0.20)	(99.70, 0.01)	(99.50, 0.02)	(97.80, 0.05)	
cMPUF	64	4	1/2	(49.80, 2.27)	(49.99, 0.23)	(83.76, 0.63)	(71.80, 0.89)	(52.64, 0.43)	12
			1/20	(49.86, 1.77)	(50.00, 0.16)	(98.69, 0.06)	(97.82, 0.08)	(94.09, 0.25)	
			1/80	(49.86, 1.76)	(50.00, 0.16)	(99.69, 0.01)	(99.48, 0.02)	(98.71, 0.05)	
		5	1/2	(50.25, 2.09)	(50.00, 0.18)	(78.82, 0.70)	(64.85, 1.15)	(50.13, 0.08)	21
			1/20	(50.23, 1.64)	(50.00, 0.13)	(98.10, 0.08)	(96.71, 0.17)	(87.09, 0.29)	
			1/80	(50.23, 1.62)	(50.00, 0.13)	(99.55, 0.02)	(99.22, 0.04)	(97.12, 0.08)	
	128	4	1/2	(49.65, 1.83)	(50.00, 0.14)	(83.78, 0.43)	(71.83, 0.98)	(52.66, 0.23)	12
			1/20	(49.75, 1.46)	(50.00, 0.10)	(98.69, 0.04)	(97.82, 0.11)	(94.12, 0.17)	
			1/80	(49.74, 1.45)	(50.00, 0.10)	(99.70, 0.01)	(99.49, 0.03)	(98.72, 0.04)	
		5	1/2	(49.89, 1.34)	(50.00, 0.11)	(78.84, 0.40)	(64.85, 0.64)	(50.12, 0.08)	21
			1/20	(49.90, 1.03)	(50.00, 0.10)	(98.11, 0.05)	(96.73, 0.10)	(87.05, 0.20)	
			1/80	(49.91, 1.02)	(50.00, 0.10)	(99.55, 0.01)	(99.23, 0.03)	(97.11, 0.05)	
rMPUF	64	3	1/2	(50.01, 4.40)	(49.88, 1.02)	(85.10, 0.41)	(78.67, 0.78)	(60.25, 0.45)	15
			1/20	(50.04, 3.80)	(49.95, 0.69)	(98.67, 0.07)	(98.02, 0.11)	(92.93, 0.17)	
			1/80	(50.02, 3.81)	(49.95, 0.69)	(99.68, 0.02)	(99.50, 0.04)	(98.14, 0.08)	
	128	3	1/2	(49.60, 2.24)	(49.98, 0.62)	(85.02, 0.32)	(78.74, 0.57)	(60.28, 0.35)	
			1/20	(49.58, 1.98)	(49.99, 0.41)	(98.66, 0.05)	(98.01, 0.10)	(92.95, 0.14)	
			1/80	(49.59, 1.97)	(49.99, 0.41)	(99.68, 0.02)	(99.50, 0.03)	(98.13, 0.05)	

$^\dagger x$ represents the number of APUFs used in the MPUF or rMPUF
$^\star \sigma_{\text{noise}} = \alpha\sigma$, where $0 \leq \alpha \leq 1$

authentication protocol. Since rMPUF overcomes the limitations of XOR-PUF, this proves to be a much better alternative to put to practice.

8.4.1 Design of PUF-Based Protocols

One of the major security challenges in IoT framework is the authentication and key management of potentially billions of heterogeneous devices deployed in the network. We try to address this problem and provide a lightweight and secure solution amalgamating the concepts of PUF, Identity-Based Encryption (IBE), and key hash function. Conventional PUF-based protocols are accompanied with the challenge of storing and securing the secret CRP database at the verifier end. In order to offload storage requirement from verifier and to eliminate risk of getting CRP database compromised, we propose a new mechanism wherein we store just a single key in the Non-Volatile Memory of verifier for authentication of all prover nodes under it using a key-ed hash function, thus reducing the space required drastically. This way it would be easier to protect a single key instead of securing a whole CRP database. Additionally instead of using CRP database directly we generate a new security association information between prover and verifier that hides the correlation between the challenge and response of the PUF and can be stored as public information.

We have used this protocol on a video surveillance camera and tested its effect in protecting the device against "man-in-the-middle" and "replay" attacks. Figure 8.12a, b presents a

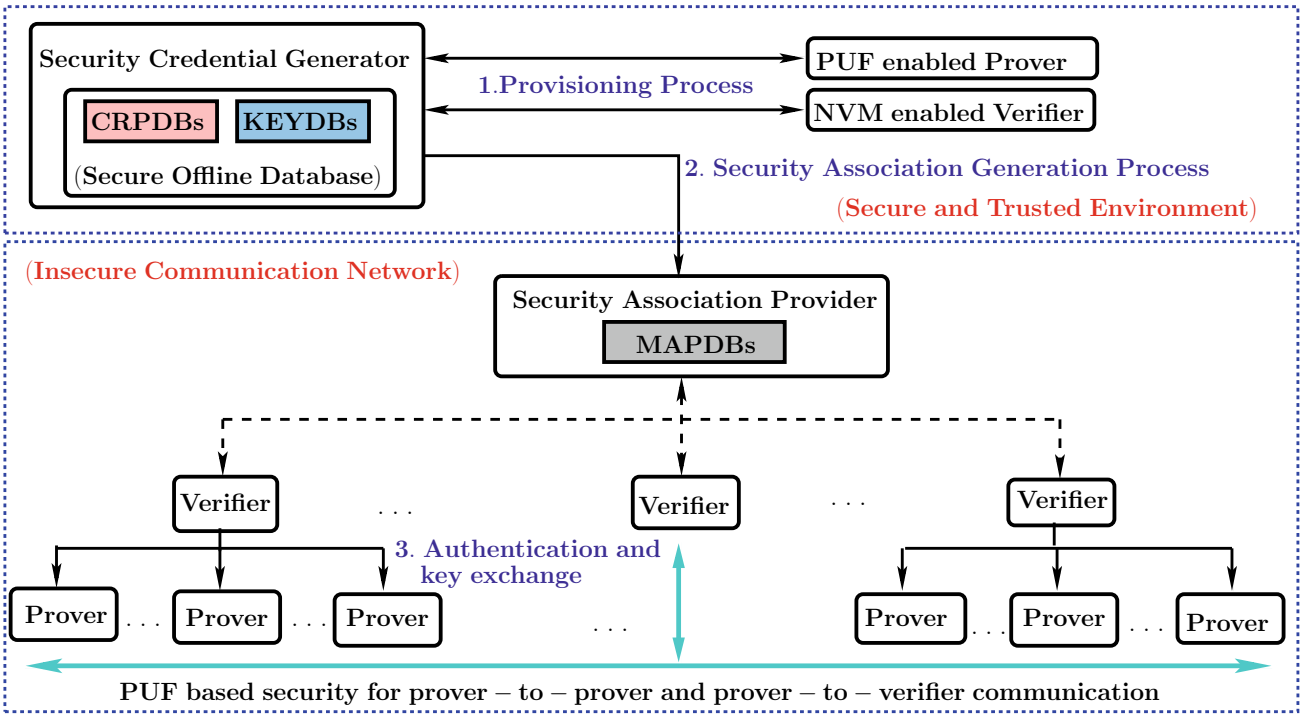


Fig. 8.11 Hierarchical IoT architecture and the proposed secure communication mechanism [19]

practical attack on a video surveillance camera which is successful in the absence of a PUF-based authentication mechanism while Fig. 8.12c, d shows the prevention of the same attack in the presence of our proposed authentication mechanism. The attack was conducted on a Logitech HD UVC camera, which is connected to Intel Edison Board via USB interface to form an IoT node. An mpg-streamer software is run on Edison Board to capture video and send it to the receiver via WiFi, which is then displayed in a web browser using the IP Address of Edison Board. Using hacking tools, the attacker can break the video streaming from the IoT node end and stream pre-captured video from attackers end, hence compromising security. In the presence of our protocol, when an attacker tries to de-authenticate a valid IoT source node and authenticate a malicious node, the receiver will ask for re-authentication, which would require the correct PUF instance, hence leading to a failure. The power consumption of the circuit is reported to be as low as 0.044 W, which makes it suitable for the IoT framework.

8.5 Micro-architectural Attacks and Countermeasures

Micro-architectural features leave a footprint in the processor which is often captured by side channels. In recent microprocessors, various architectural components are incorporated in the system to improve the system performance and these are emerging as new sources of side-channel leakage. Recent micro-architectural attacks named Spectre and Meltdown have taken the world by storm by uncovering two processor vulnerabilities. These vulnerabilities were there in the processor design for decades and uses architectural constructs such as the branch predictors and speculative execution optimization to realize a practical breach of security. We illustrate that the cache memory execution footprints when analyzed with the knowledge of the underlying cache memory structure and characteristics lead to leaking the secret key bytes of block ciphers. On the other hand, when asymmetric-key cryptographic algorithms are implemented and the branch-predictor hardware is monitored, the ciphers are subjected to side-channel attack because of their key-dependent input sequences. We also demonstrate a software-driven fault attack using row-hammer to induce bit flips in the cryptographic secret located in DRAM, by repeated charging and discharging of the memory elements located in the same DRAM bank as the cryptographic secret. The most prominent attack algorithms conceived in SEAL using the micro-architectural primitives are listed below in more details.

8.5.1 Cache Timing Attack on Clefia

We start with the work presented in [20], which discusses the performance of cache timing attacks on Clefia. Clefia has a generalized Feistel structure and unlike other Feistel struc-

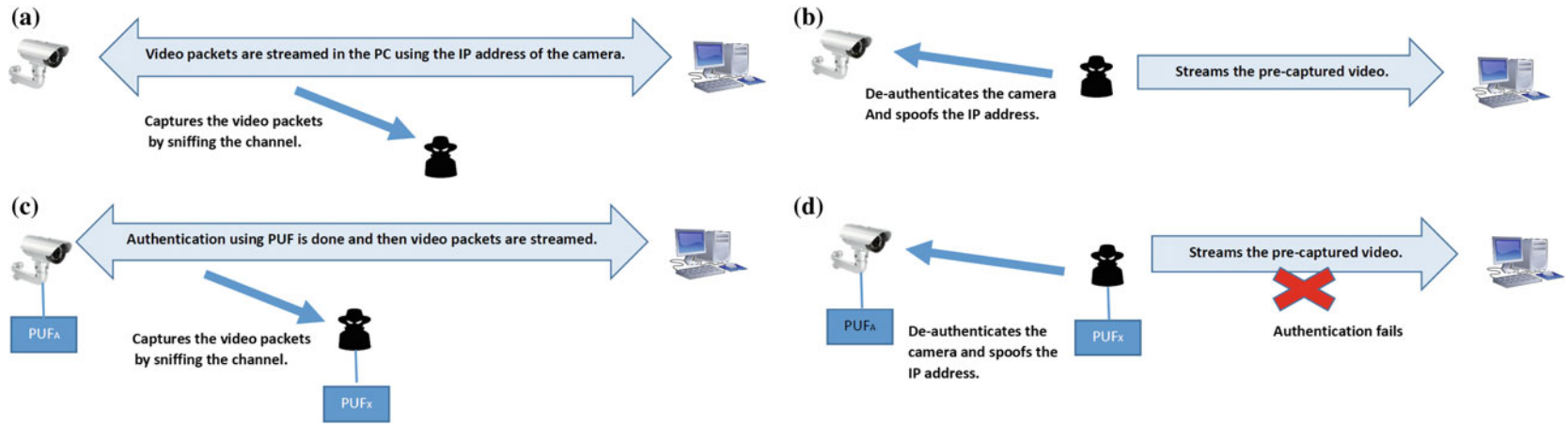


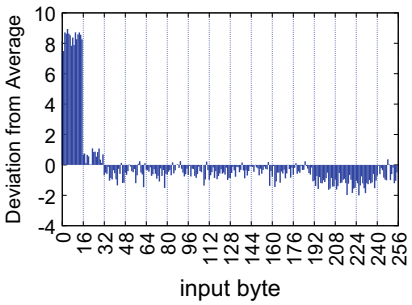
Fig. 8.12 Attack on video surveillance system and protection against it: **a** and **b** show the successful attack in the absence of PUF-based authentication mechanism, while **c** and **d** show the prevention of the attack in the presence of the proposed PUF-based authentication system [19]

tures, it has been implemented using small tables. This was the first attack of its kind to propose a full-scale cache timing attack on ciphers with small tables. The attack uses the fact that parallelization and pipelining of memory accesses can only be performed within a single round of a cipher, and not across rounds.

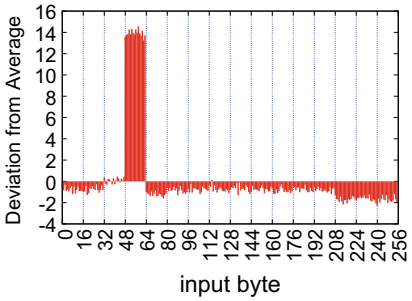
Among the several design challenges which are supported by today's microprocessors in order to reduce the miss penalty the most important are speculative loading, out-of-order loading, prefetching, parallelization, and overlapping. Speculative loading enables data to be loaded into the cache memory before preceding branching instructions are evaluated and resolved. Prefetching is performed when the hardware prefetcher detects a sequence of memory accesses in a specific order. In out-of-order loading, the processor accesses memory elements in a sequence which is not strictly specified by the program. For a block cipher, speculative loading and hardware prefetching has no effect on the execution time because the key-dependent load operations are random in nature. Additionally cache misses are usually handled out of order. In block ciphers like Clefia, outputs from one round are used in the next, while operations within a round are accessed independent of each other. This implies that memory accesses within a round of Clefia can be performed out of order, but adjacent rounds waits for the previous round to complete and thus follows a sequence.

The attack, demonstrated as follows, modifies Bernstein's cache timing attack [22] with the observation that in the first round the cache timing profile for the ith byte of the key k_i can be affected only by those plaintexts which access the same table because accesses to different tables does not contribute to timing deviations in respective timing profiles. Cache accesses in tables other than the table used by the key byte k_i causes unrequired deviations and thus increases the error in the profile for k_i. This was illustrated in our work to generate more accurate timing profiles. The timing measurement was also improved in compared to the measurement techniques there in the literature. Using the `rdtsc` instruction for timing measurement is known to have errors in measurement due to the out-of-order execution in the pipeline. In our timing measurement, the `cpuid` instruction is invoked before the `rdtsc` to flush the pipeline thus reducing errors [21].

Figure 8.14 shows two accesses to the same s-box table with indices $(in_0 \oplus k_0)$ and $(in_1 \oplus k_1)$, where in_0 and in_1 are the inputs and k_0 and k_1 are the key. Cache hits occur when $(in_1 \oplus k_1)$ fall in the same cache line as $(in_0 \oplus k_0)$, in all other cases cache misses occur. A typical illustration on the timing profiles of the known and unknown keys are provided in Fig. 8.13. The profiled cache timing attack has 3



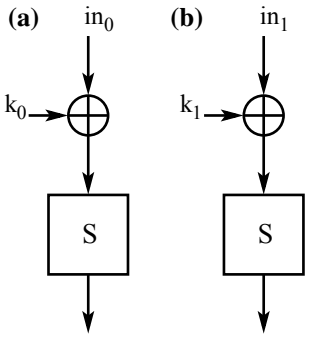
(a) Timing Profile for the Unknown key bytes



(b) Timing Profile for the Known key bytes

Fig. 8.13 Timing profiles for all possible values of [21]

Fig. 8.14 Simple S-box Lookup structure



phases: learning, attack, and analysis phase. During the *learning phase*, the adversary uses a known key (or an exact replica of the attack platform) to build a *timing profile* for each key byte. The profile has on the x-axis, all possible values corresponding to the plaintext byte p_{15}, and on the y-axis the average encryption time corresponding to when p_{15} is fixed at a certain value and the remaining input bytes varied randomly. Note that Fig. 8.13b shows the deviation from average encryption time, instead of the actual encryption time. The timing profile is built with 2^{24} encryptions and called the *template* in this phase of the attack. During the *attack phase*, the adversary builds a similar timing profile, only this time for the unknown key byte. Such a profile is shown in Fig. 8.13a. It can be noted that this profile is very similar to the template in Fig. 8.13b, except for a shift, which occurs due to the *equal*

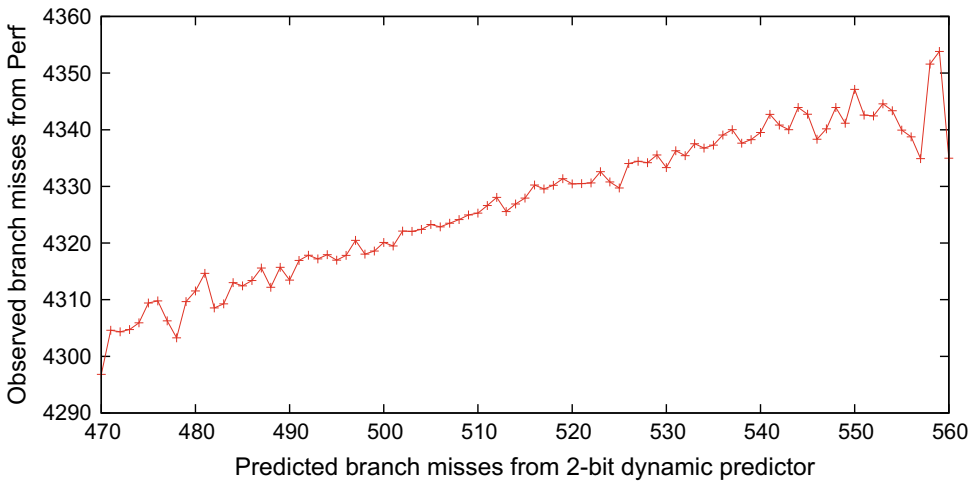


Fig. 8.15 Variation of branch misses from performance counters with increase in branch miss from 2-bit predictor algorithm [23]

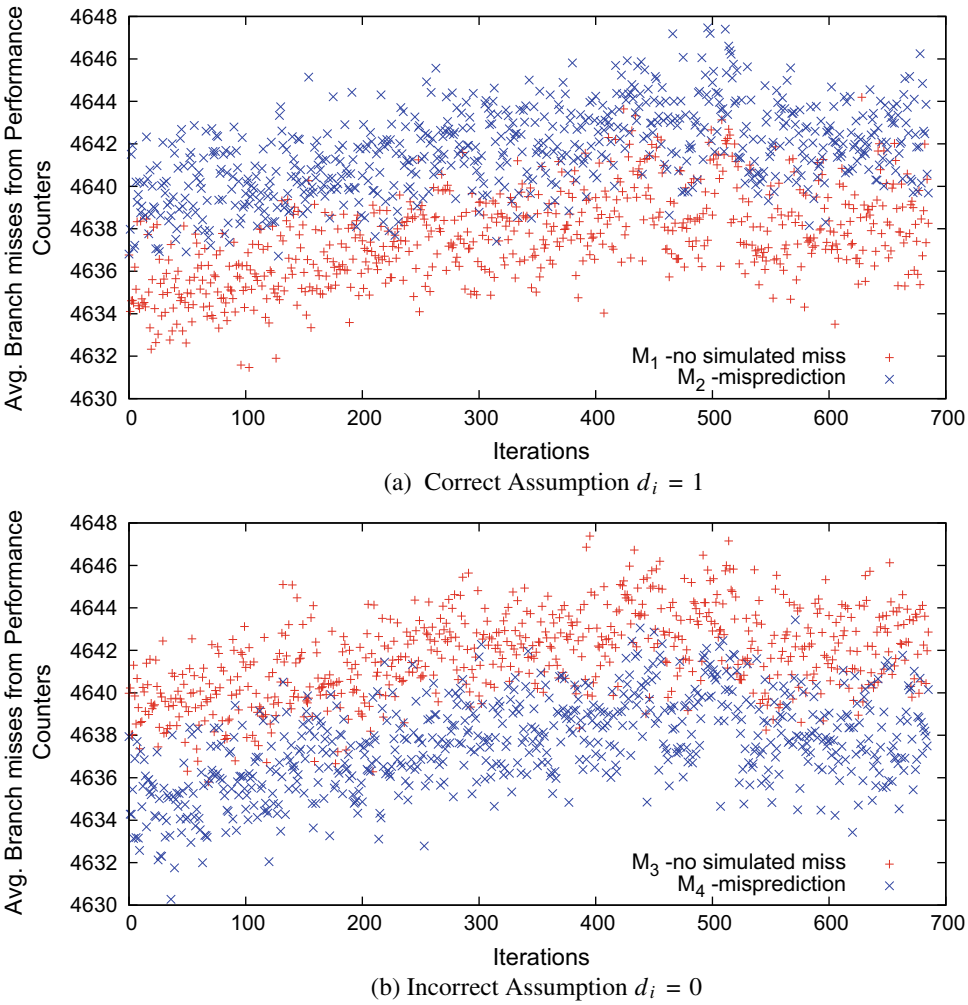
images under different sub-keys (EIS) property of the cipher. It is the EIS property that results in the leakage of information about the secret key. During the *analysis phase*, this shift is determined using correlation techniques in order to retrieve the unknown key byte. In a similar manner, timing profiles constructed for other bytes can be used to determine other parts of the key. Our findings show that 121 bits of the 128-bit key can be revealed in $2^{26.64}$ Clefia encryptions on an Intel Core 2 Duo machine.

8.5.2 Branch Misprediction Attack

Asymmetric-key cryptographic algorithms when implemented on systems with branch predictors, are subjected to side-channel attacks exploiting the deterministic branch-predictor behavior due to their key-dependent input sequences. In our work [23], we show that branch predictors can also leak information through the hardware performance monitors. We construct an iterative attack which target the key bits of 1024-bit RSA, where in the offline phase, the system's underlying branch predictor is approximated by a theoretical predictor in the literature. Figure 8.15 illustrates that the 2-bit bimodal predictors bear a direct correlation to the underlying hardware predictor in Intel systems. Subsimulations are performed to classify the message space into distinct partitions based on the event branch misprediction and the target key bit value. In online phase, we ascertain the secret key bit using branch mispredictions obtained from the hardware performance monitors which reflect the information of branch miss due to the underlying predictor hardware. We provide an analysis to justify that the probability of success to guess the ith bit correctly is equivalent to the extent of resemblance of theoretical predictor behavior to the underlying system predictor hardware.

Figure 8.16 shows the correct and incorrect separations for all 4 sets (separated by simulations over two-level adaptive predictor) for the randomly chosen 548th bit location of the target key-stream. Define M_1 as the set which does not cause a miss during multiplication of $(i+1)$th squaring when secret bit $d_i = 1$. Likewise M_2 corresponds to set for a miss. Figure 8.16a plots average branch misses observed from performance counters for each elements in set M_1 and M_2 (each set having L = 1000 elements) and the experiment is repeated over I = 1000 iterations in order to check the consistency of the output. It is evident from the figure that in most of the iterations the average branch miss for set M_2 is more than the branch misses for set M_1 (as expected). We define M_3 and M_4 to be similar sets as M_1 and M_2 which corresponds to $d_i = 0$. Fig. 8.16b plots average branch misses observed from performance counters for each elements in set M_3 and M_4. But we observe an incorrect separation as in most of this case, ciphertexts in set M_4 is having lesser branch misses than in set M_3 which is incorrect since theoretically it should be the reverse. Thus from these two figures, the correct exponent can be easily identified showing correct difference in branch misses.

We also fine-tune this original attack strategy in [24] such that this alternative strategy can retrieve the secret bits much more efficiently and requires much lesser number of inputs. The experimental validations of both the attack strategies are illustrated on public-key cryptographic algorithms as RSA and ECC on several Intel platforms show significant success revealing the secret exponent and scalar value. We also extend our attack to the RSA-OAEP randomized padding procedure where we target the decryption phase of the implementation and the branch miss side-channel information of the entire decoding procedure can be successfully exploited to reveal the secret exponent. What make these results more relevant is the fact that protections which fuzz the timing channels are not sufficient to thwart these attacks, and present performance counters as a distinct side channel which attracts attention to



(a) Correct Assumption $d_i = 1$

(b) Incorrect Assumption $d_i = 0$

Fig. 8.16 Branch misses from HPCs on square and multiply correctly identifies secret bit $d_i = 1$, ciphertext set partitioned by simulated misses of two-level adaptive predictor [23]

further systematic research to develop systems that are inherently secure.

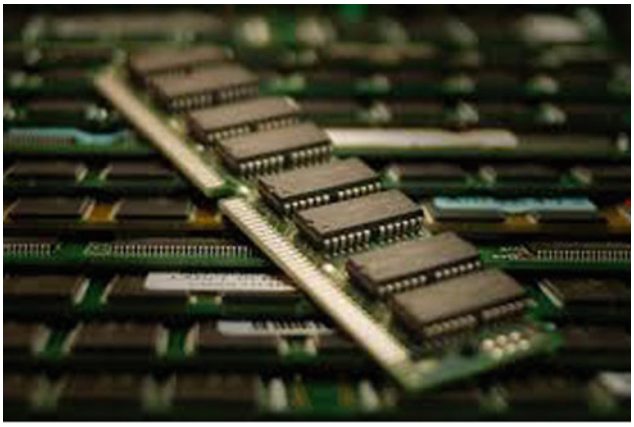
8.5.3 Software-Driven Fault Attack Using Row-Hammer

Row-hammer is a term coined for a disturbance in DRAM resulting in bit flips due to repeated charging and discharging of DRAM rows in a particular bank. This process is hardware dependent and highly probabilistic but repeatable phenomenon, thus localizing and inducing fault in a specific location of memory is a hard problem till date. Figure 8.17 shows the DRAM architecture and row-hammer is flipping of bits in the DRAM cells neighboring row of heavily accessed DRAM rows.

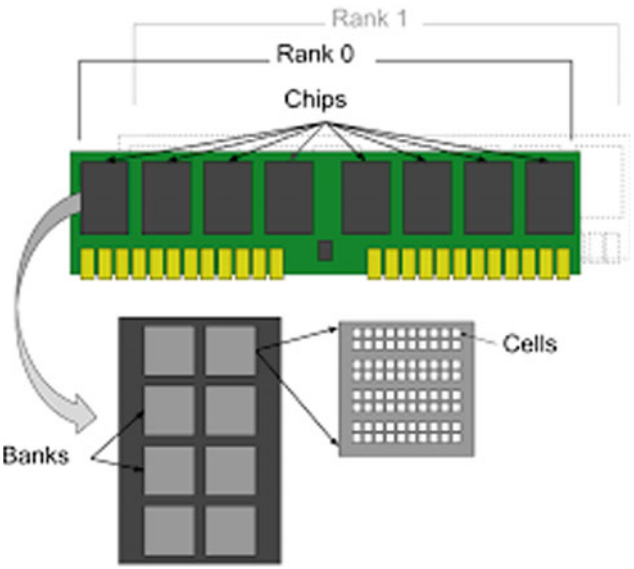
This work brings into practise a methodology to combine timing analysis to perform row-hammering in a controlled manner to create bit flips in cryptographic keys which are stored in memory [25]. The attack would require only user-level privilege for Linux kernel versions before 4.0 and is unaware of the memory location of the key. An intelligent combination of timing Prime + Probe attack and row-buffer collision is shown to induce bit flip faults in a 1024-bit RSA key on modern processors using realistic number of hammering attempts. This demonstrates the feasibility of fault analysis of ciphers using purely software means on commercial x86 architectures, which to the best of our knowledge has not been reported earlier.

We combine knowledge of reverse engineering of LLC slice and DRAM addressing with timing side channel to determine the bank in which secret resides. The overall idea of the attack involves three steps. The attacker successfully identifies an *eviction set* by following the series of steps as illustrated in Fig. 8.18. The eviction set comprises a set of data elements which map to the same cache set and slice as that of the secret exponent and we identify such set by timing analysis using *Prime + Probe* methodology. This set essentially behaves as an alternative to `clflush` statement of x86 instruction set.

The attacker now observes timing measurements to DRAM access by following the series of steps in Fig. 8.19 to determine



(a) A typical DRAM card



(b) DRAM Architecture

Fig. 8.17 Typical DRAM hierarchy

the DRAM bank in which the secret resides in. The variation in timing is observed due to the row-buffer conflict forced by the adversary.

This leads to repeated opening and closing of rows in DRAM banks inducing bit flips by repeated row activation in the particular bank where the secret is residing. We precisely trigger row-hammer to address in the same bank as the secret. This increases probability of bit flip in the secret exponent and the novelty of our work is that we provide series of steps to improve the controllability of fault induction.

A statistic over observations of bit flips in respective banks is reported in Fig. 8.20. The bar graph shows the number of bit flip instances we were able to observe for respective banks of a single Dual In-line Memory Module (DIMM).

8.5.4 Detection of These Attacks

There are various state-of-the-art countermeasures in the literature to prevent different types of micro-architectural attacks. Some of these countermeasures are implemented by significantly changing the hardware [26], obscuring timing information [27], etc., thereby incurring the cost of extra overhead for their implementations. The severe overhead cost cuts down the performance of the system and increases the energy consumption of the device manifold. A well-established fact is that modern computers will exhibit information leakages, and we need to develop our own defence amid the presence of these leakage avenues, and that too with a low implementation overhead.

Micro-architectural side-channel attacks primarily focus on the execution of the target encryption algorithm and its impact on the behavior of shared hardware resources like CPU-cache, branch-predictor hardware, Dynamic Random Access Memory (DRAM), etc. These attacks on the encryption algorithms manifest the effect of these shared hardware resources in the presence of a concurrently running spy process, which in turn continually monitors the timing or the hardware performance counter statistics of the target secret execution. As a result of these continuous monitoring, the spy processes leave a footprint on the hardware resources. Our objective is to devise a detection module which studies the behavior of these attack algorithms using the low-level hardware events through HPCs. In the presence of these attacks, abnormalities in the number of performance counter events are abrupt in comparison to the normal system execution. In Fig. 8.21, we can observe the distributions of different hardware resources in the presence of various micro-architectural attacks and it is clear from the figure that the micro-architectural attacks work as an anomaly in comparison to the normal behavior of a target system.

Our first approach [28] is to generate a significant volume of low-level data by profiling the hardware performance counters at a granular measurement. We can observe from Fig. 8.21 that attack behaviors are different from the normal process behavior for specific hardware events based on their types. Thus we analyze the cause of abnormality, if any, for the running processes and categorize them into appropriate classes using a pretrained classifier. The result of this phase would help us to comprehend the type of the possible attack process, i.e., whether a cache-based attack or a branch-based attack and so on. We develop a basic template for the target system that we intend to protect against the micro-architectural side-channel attacks, and the process of obtaining information from the system and training a classifier is shown in Fig. 8.22a. However, certifying that a process is actually an attack process and not any high computation process needs further analysis with the performance counter values. The existence of false positives, having an approximately equal

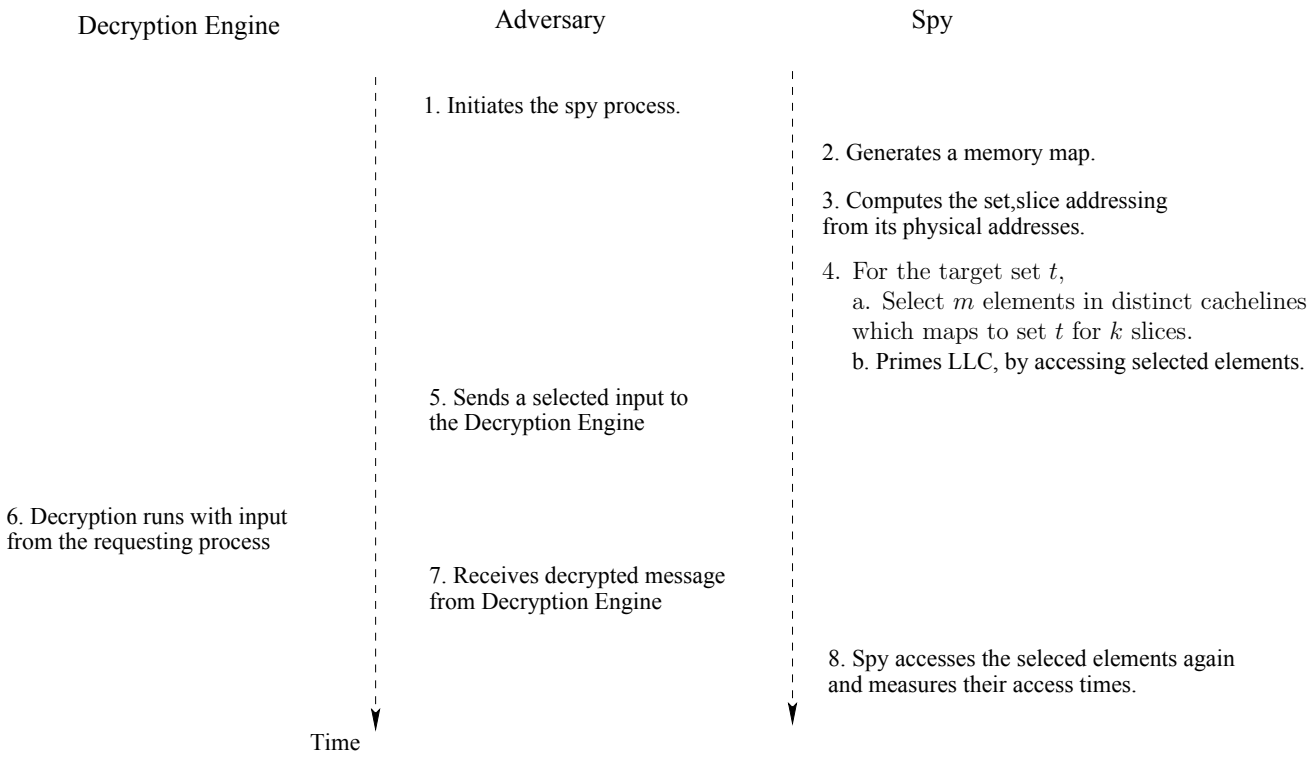


Fig. 8.18 Steps to determine cache sets shared by secret exponent [25]

effect on the performance counter values as that of the targeted attack processes, may confuse the classification phase. Here we introduce the second step of our detection mechanism which removes these false positives by correlating respective hardware events (i.e., cache events for cache-based attacks, branch events for branch-based attacks, etc.) with the secret key of encryption. A true side-channel attack has the highest correlation with the secret key as it needs to repeatedly access the encryption algorithm for retrieving the secret key using statistical evaluations. In order to measure the correlation of the execution trace of an unknown process with the secret key, we collect the HPC values from the target system executing the target encryption algorithm for different plaintexts and a *fixed* secret key. The data collection process is shown in Fig. 8.22b. We store the collected trace for later considering it to measure correlation with an unknown process during the online phase.

In the online phase, we continuously monitor the target system executing the target encryption algorithm and examine whether the system is in a safe state or under the threat of any side-channel attack. The online phase utilizes the classifier and the stored traces prepared during the offline phase. The procedure of the online mode of operation is shown in Fig. 8.23. Based on the category of anomaly obtained from the classifier, the hardware trace of the encryption algorithm is chosen from the stored traces related to the appropriate performance counter events. The correlation value of this trace with the trace obtained by monitoring the anomalous process is then calculated using the Dynamic Time Warping (DTW) method. A low correlation value signifies that the system is in a safe state and the anomalous process is just a high computation program which has no relation to the secret key of encryption, whereas, a high correlation value determines the existence of a possible side-channel attack in the background.

Figure 8.24a shows different nonlinear decision boundaries learned by a Random Forest Classifier from the data obtained by monitoring the hardware events during the offline phase. We can observe that various anomaly models can be well separated from each other based on the acquired data. Figure 8.24b shows the detection of *Cache Timing Attack* on Clefia [29] using the DTW correlation. We can observe from the figure that the attack process has low DTW value with the stored trace in comparison to a benign *Firefox* application, thereby removing the false positives obtained from the classifier.

8.6 Hardware Security to Accelerate Cloud Cryptosystems

Recent progress of cloud-based computation and storage infrastructure have fueled the demand for stronger security for users' data on the cloud. Traditional encryption schemes allow encrypting data to prevent unauthorized access from

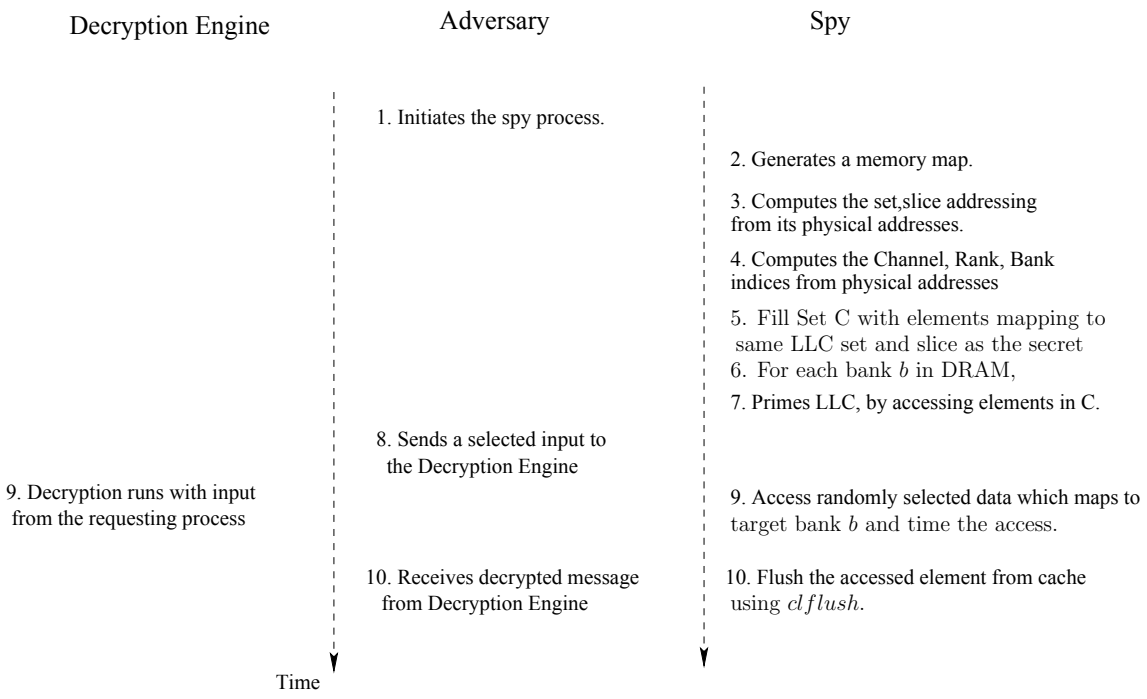


Fig. 8.19 Steps to determine the DRAM bank in which secret maps [25]

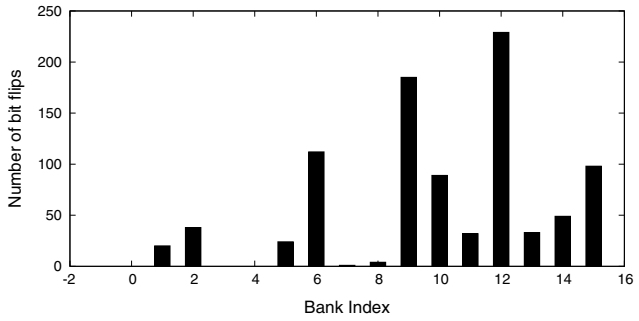


Fig. 8.20 Number of bit flips observed in all banks of a single DIMM [25]

adversaries. However, these schemes fail to provide any extended functionalities to perform particular operations or to evaluate functions over the encrypted data. Searching over encrypted data, performing arithmetic computations over encrypted data, and pattern matching are some of the prominent operations which a cloud infrastructure should provide, but implementing these for an encrypted database, generally in encrypted domain, is a challenging problem in cryptographic research. Currently encrypted domain computation has turned up to be an attractive topic for researchers, development of new generation of functional encryption schemes and homomorphic encryption schemes allow to implement the aforementioned functionalities or operations over encrypted data. Among these, Searchable Symmetric Encryption (SSE) schemes are of special interest, giving the users the ability to search for keyword(s) over encrypted data with restricted access. Traditional SSE schemes employ complex public key based constructs to facilitate the search functionality, and most of those are limited to single keyword search only. This creates a bottleneck for practical implementations preventing deployment at a large scale; primarily for the slower public key based constructions which curtail the throughput of the implementations.

We developed a highly efficient and scalable SSE scheme [30] based on the symmetric key primitives only, with support for multiple conjunctive keyword search. This scheme, referred to as the Hidden Cross Tags (HXT) protocol offers multi-fold improvement over the existing SSE schemes and outperforms other practical implementations which has been validated by software implementation. This scheme incorporates a novel symmetric key based Hidden Vector Encryption (HVE) scheme in the construction. Originally, the Hidden Vector Encryption schemes employ complex Bilinear Map based constructions, which are slow and inefficient, rendering the overall scheme unsuitable for practical use. This Symmetric key Hidden Vector Encryption (SHVE) provides an efficient and more secure symmetric key based approach to reduce the complexity of traditional HVE, thus allowing us to develop a completely symmetric key based SSE. To best of our knowledge, this is the first SSE scheme with complete symmetric key-based construction.

The HXT protocol constitutes of two top-level algorithms, the $SE.EDBSetup()$ and the $SE.EDBSearch()$, with multiple server–client interactions taking place at different

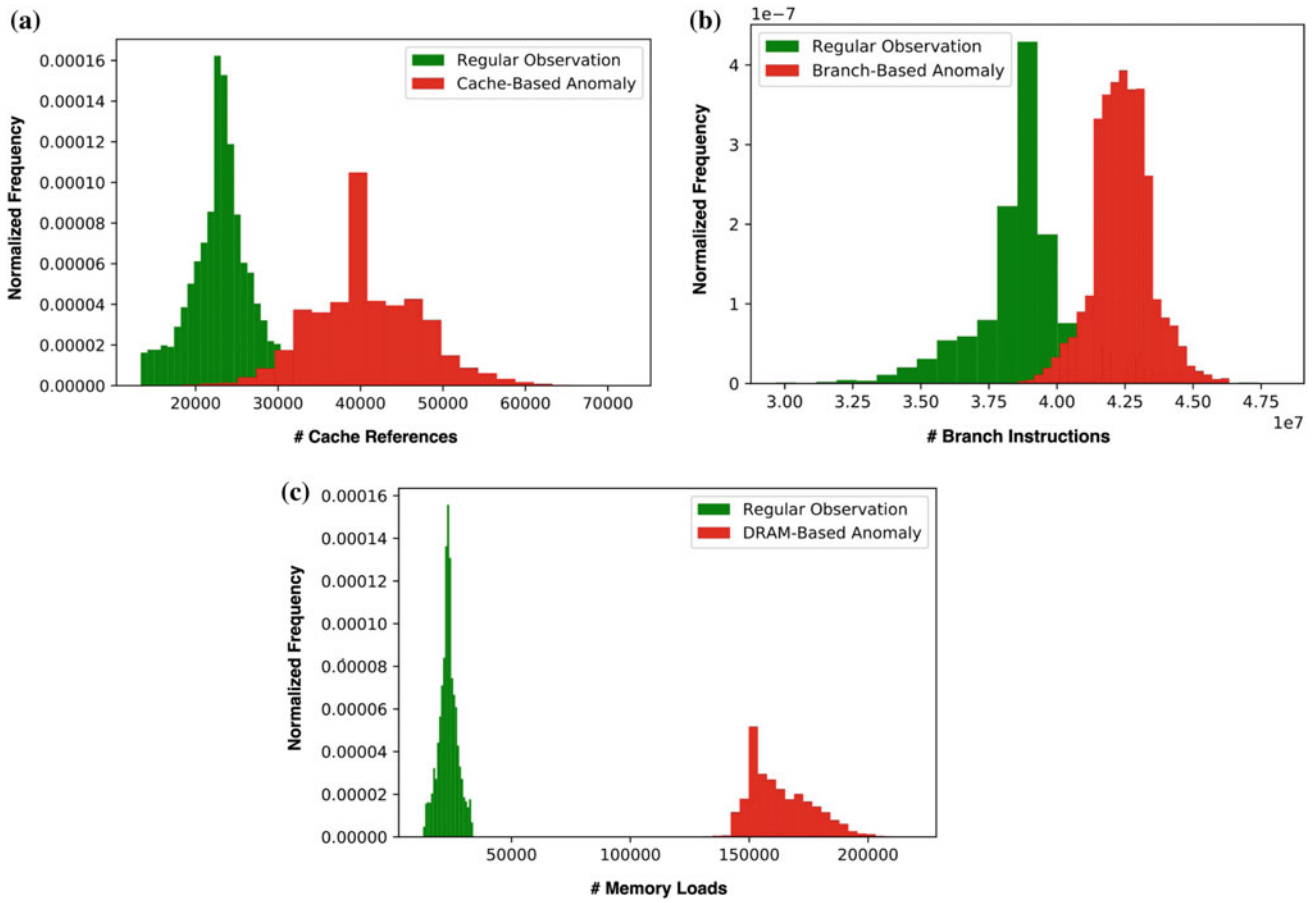


Fig. 8.21 Distinct distributions of **a** Cache-references for regular observations and cache-based anomaly, **b** Branch instructions for regular observations and branch-based anomaly, and **c** Memory-loads for regular observations and DRAM-based anomaly

phases. The $SE.EDBSetup()$ algorithm is responsible for constructing the encrypted database, setting up the system parameters and generating the necessary keys for the protocol. This setup algorithm takes the security parameter λ and the document database DB as input and generates the encrypted database EDB and a master key mk. EDB contains two separate encrypted databases $EDB1$ and $EDB2$, and the master key mk is a collection of several other keys used in the construction. The $SE.EDBSearch()$ algorithm takes the master key mk, query string $q = w_1, w_2, w_3, \ldots, w_n$ and EDB as input and returns the list (encrypted) of document identifiers for the query q.

Additionally, HXT scheme has two other modules, the $TSet$ and the $SHVE$. The algorithms associated with $TSet$ are described below.

- $TSet.Setup(T)$: This algorithm is responsible for the generation of the actual encrypted database $TSet$ from keyword database T, and generation of the corresponding key K_T.
- $TSet.GetTag(K_T, w)$: This algorithm generates a tag $stag$ for a keyword w using the corresponding key K_T.
- $TSet.Retrieve(TSet, stag)$: This algorithm retrieves the document identifier list t corresponding to the keyword w associated with $stag$.

The other major sub-module SHVE has been discussed later in detail. The algorithms belonging to the sub-modules $TSet$ and $SHVE$ are used in different phases of $SE.EDBSetup()$ and $SE.EDBSearch()$. An illustration depicting the server–client interaction at different phases of SSE and the respective algorithms is given in Fig. 8.25. For the complete description of the algorithms $SE.EDBSetup()$ and $SE.EDBSearch()$, the readers may refer to the original paper.

The drastic performance improvement of HXT is primarily due to the SHVE construction. The complete description of SHVE construction is provided below.

Denote a finite set of attributes as Σ and define $\Sigma_* = \Sigma \cup \{*\}$ where $*$ is a wildcard symbol. Define a family of predicates as $\mathrm{P}^{SHVE}: \Sigma \longrightarrow \{0, 1\}$, where for a particular $\mathrm{v} = \{v_1, v_2, v_3, \ldots, v_m\}$, we have a predicate $P_{\mathrm{v}}^{SHVE} \in \mathrm{P}^{SHVE}$ satisfying

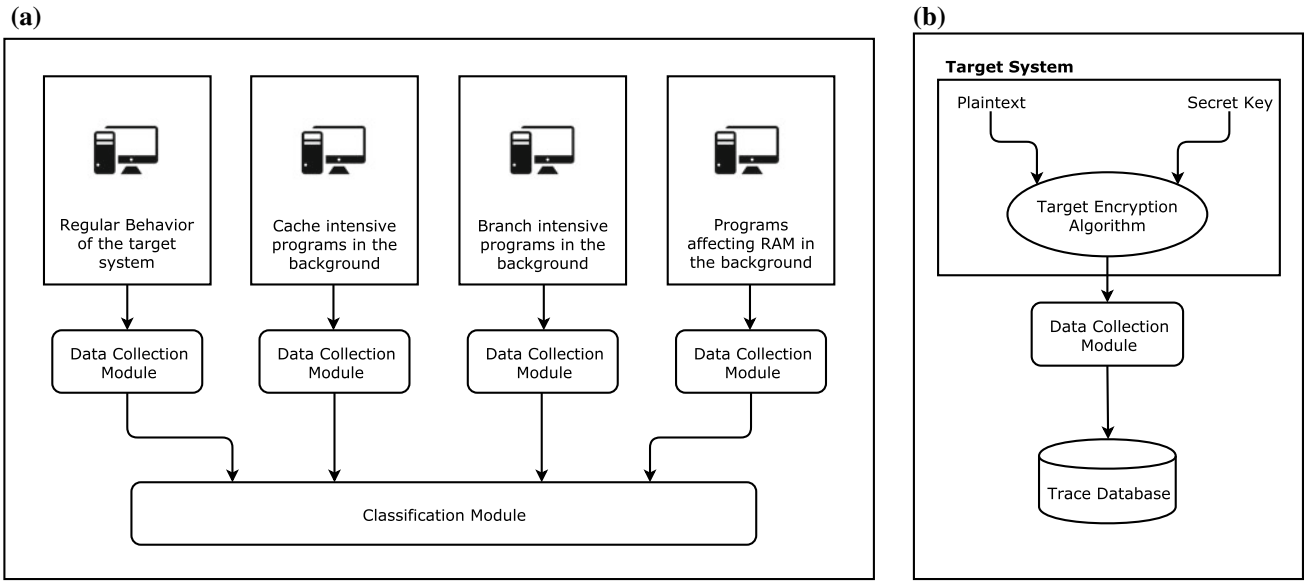


Fig. 8.22 **Offline Phase**: **a** Collection of data for *regular behavior* of the *target system* as well as for different types of *micro-architecture intensive* (like cache-based, branch-based, RAM-based, etc.) programs executed in the background of the target system. **b** Data collection and building a template for a *fixed* secret key considering *target encryption algorithm* (like Clefia, AES, RSA, etc.) for the *target system*

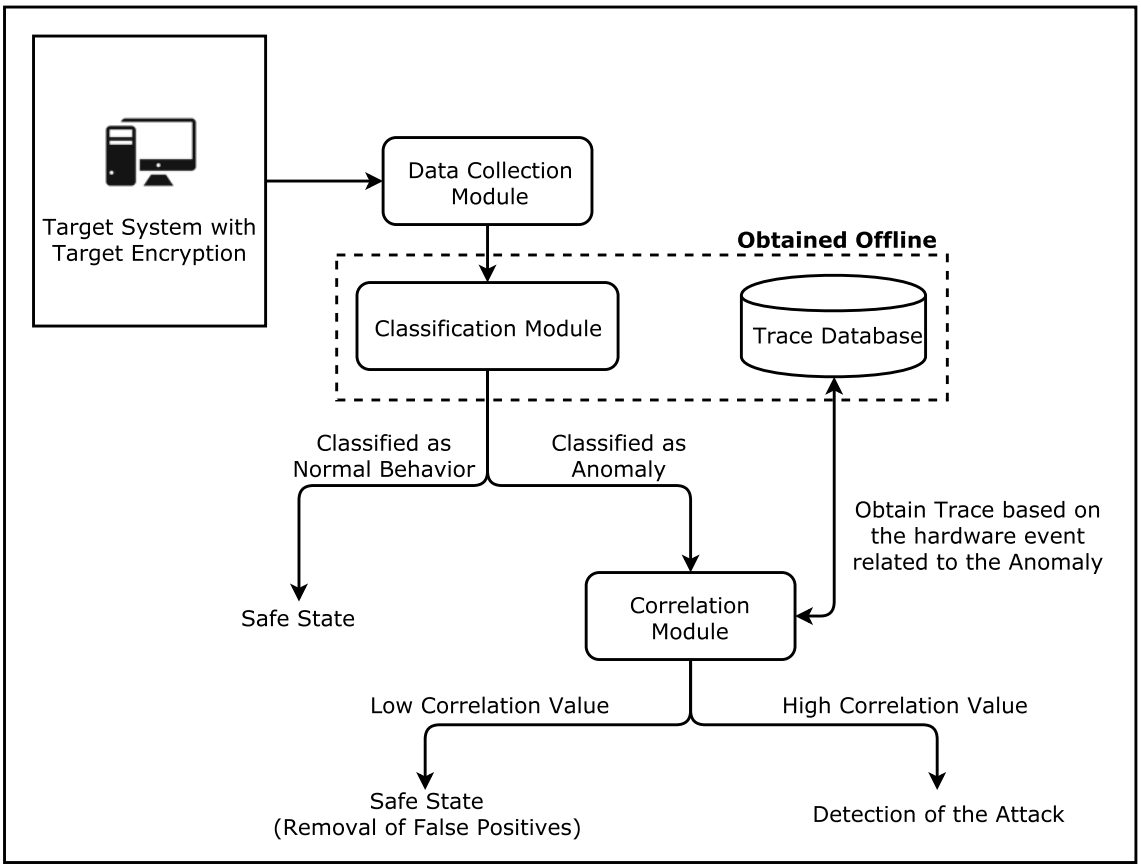


Fig. 8.23 **Online Phase**: Collection of data from the *target system* executing the *target encryption algorithm* and determining whether the system is in *Safe State* or *Under Attack* using the *Classification Module* and *Trace Database* obtained from the *offline* phase

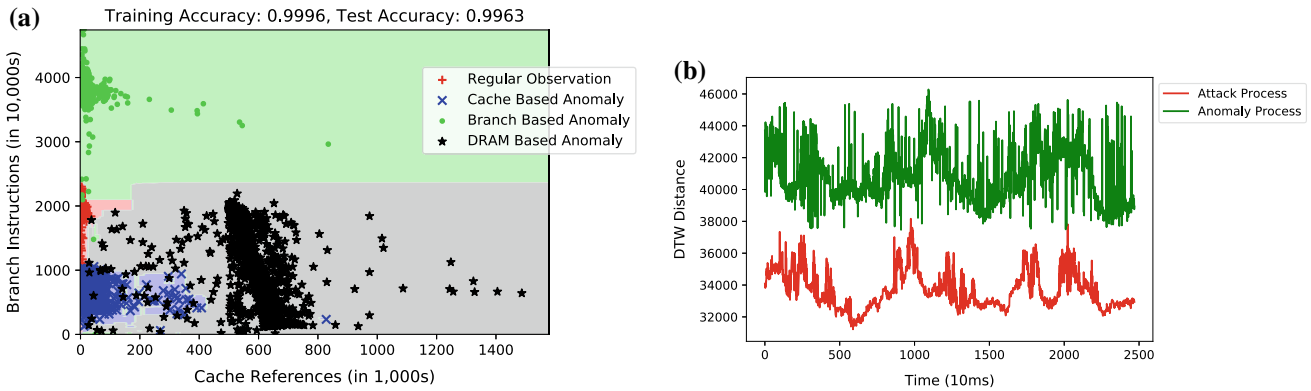


Fig. 8.24 **a** Decision boundaries random forest classifier **b** DTW distances for both cache timing attack on *clefia* (attack process) and *firefox application* (anomaly process)

$$P_{\mathrm{v}}^{SHVE}(\mathrm{x}) = \begin{cases} 1 \; \forall 1 \; \leq i \leq m \; (v_i = x_i \text{ or } v_i = *), \\ 0 \text{ otherwise} \end{cases}$$

and $\mathrm{x} = \{x_1, x_2, x_3, \ldots, x_m\} \in \Sigma^m$.

Two cryptographic primitives, Pseudorandom Function (PRF) and a IND-CPA secure symmetric key encryption scheme $Sym.Enc$ are used in this construction. The PRF is defined as the mapping $F_0 : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda+\log\lambda} \rightarrow \{0,1\}^{\lambda+\log\lambda}$ and the symmetric key encryption has both the key space and the plaintext space as $\{0,1\}^{\lambda+\log\lambda}$. The SHVE has the following algorithmic procedures:

- $SHVE.Setup(1^{\lambda})$: Security parameter λ is the input for this algorithm and the algorithm samples the master secret key $msk \longleftarrow \{0,1\}^{\lambda}$. This also defines the payload message space as $M = \{True\}$ and outputs (msk, M)
- $SHVE.KeyGen(msk, \mathrm{v} \in \Sigma_*^m)$: This function takes msk and a predicate vector $\mathrm{v} = \{v_1, v_2, v_3, \ldots, v_m\}$ as input. Denote $S = \{l_j \in [m] | v_{l_j} \neq *\}$ and the locations are $l_1 < l_2 < l_3 \cdots < l_{|S|}$. Randomly sample $k_1, k_2, k_3, \ldots, k_{|S|}$ from $\{0,1\}^{\lambda+\log\lambda}$ and perform the following steps.

$$d_0 = \oplus_{j \in [|S|]} (F_0(msk, v_{l_j} || l_j) \oplus k_j)$$

$$d_1 = Sym.Enc(\oplus_{j \in [|S|]} k_j, 0^{\lambda+\log\lambda})$$

  Finally, the algorithm outputs $\mathrm{s} = (d_0, d_1, S)$.
- $SHVE.Enc(msk, \mu = \text{'True'}, \mathrm{x} \in \Sigma^m)$: This algorithm takes msk, an input message μ and an index vector $\mathrm{x} = (x_0, x_1, x_2, \ldots, x_m)$ as input and sets the following.

$$c_l = F_0(msk, x_l || l) \text{ for each } l \in [m]$$

  next it outputs the ciphertext $\mathrm{c} = (\{c_l\}_{l \in [m]})$

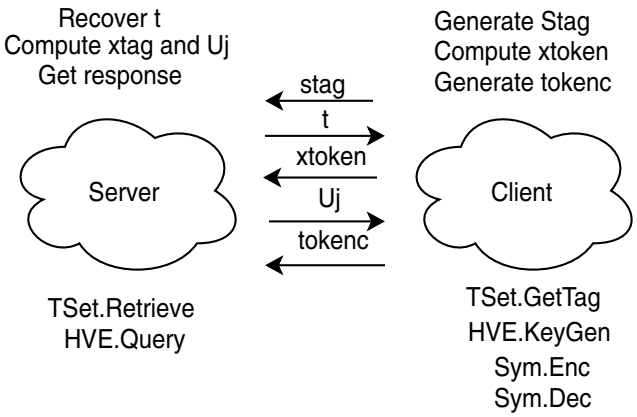


Fig. 8.25 Server–client interaction in HXT

- $SHVE.Query(\mathrm{s}, \mathrm{c})$: Previously computed s and c are passed as input to this algorithm. The algorithm computes

$$k' = (\oplus_{j \in [|S|]} c_l) \oplus d_0$$

  and then

$$\mu' = Sym.Dec(k', d_1)$$

  If $\mu' = 0^{\lambda+\log\lambda}$, it outputs "True" (the message passed to the input) else it outputs `failure` symbol $\perp$.

The performance of SHVE was evaluated in a software-based prototype implementation. From Table 8.4 it is evident that SHVE outperforms other bilinear map based HVE schemes.

The prototype implementation of HXT was evaluated on a cluster of high-performance workstations. Evaluated with three standard databases from Wikimedia Downloads of size 2.93 GB, 8.92 GB, and 60.2 GB, HXT outperforms other com-

Table 8.4 Performance comparison of SHVE with a bilinear map based HVE (IP) [31]

Scheme	KeyGen(s)	Enc(s)	Query(s)
IP	51.154	50.901	119.219
SHVE	0.172	0.162	0.004

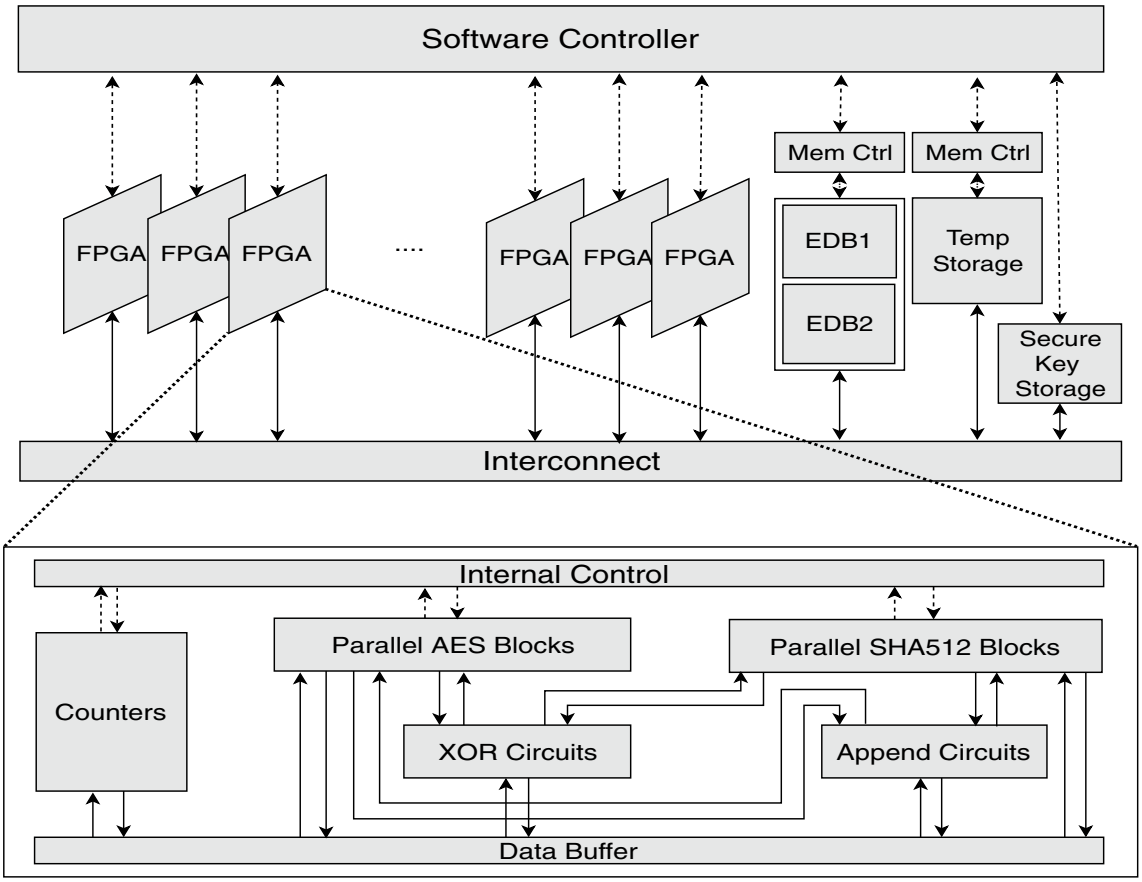


Fig. 8.26 A general architecture for parallel implementation of SSE [31]

petitive schemes by a great margin establishing superiority in this regard.

Software-based implementations provide excellent flexibility in the development of the implementations for SSE, modern-day advanced multicore processors and GPUs can speed-up an architecture for SSE to a great extent. However, a dedicated hardware-based architecture can utilize base-level customization specifically targeted for the SSE scheme. Optimized at basic block level, these hardware-accelerated implementations have the ability to perform exceptionally well compared to the software-based implementations. We proposed a dedicated parallelized hardware-based solution to accelerate the SSE scheme [31]. The HXT protocol incorporates several components which operate independently in a loop. Therefore, multiple scopes for parallelization are present in the architecture. With a meticulously designed parallelized architecture, the throughput of the system can be increased to a great extent. In our solution, we designed the architecture based on a hardware–software co-design environment, where the computation-intensive parallelized operations are accelerated by hardware, and the less computation-intensive operations are moved to the software part. The overall system is controlled by a software-based controller application running on a workstation.

Reconfigurable hardware is an attractive avenue for designers providing accelerated computation for hardware-based implementations. Flexibility, reprogrammability, and abundance of logic resources in modern reconfigurable hardware or FPGAs allow to design, test, and verify a customized design very quickly and provide the users with the ability to update and reevaluate the design within a short period. Evidently, our design is targeted for FPGA-based hardware acceleration. The computational blocks or the cryptographic primitives like AES block cipher, SHA512 hash function are the core part of each of the parallelized instances, as depicted in Fig. 8.26. Interfaced with a high-

speed data interconnect and individual node's local controller interface, these modules operate concurrently to speed-up the computation at the base level. Several lightweight designs of the standard cryptographic primitives are available in the literature, capable of delivering high throughput. Multiple instantiations of these blocks do not overload the FPGA, but increases the processing rate to a great extent. Comparing the state-of-the-art software and hardware implementations of the modules, we observe that porting AES to a dedicated hardware gains 31 times speed-up. Similarly, we observe 22 times and 13 times speed-up for SHA512 hash and elliptic curve Curve25519 point multiplication operation on reconfigurable hardware-based design.

Based on this comparison, preliminary analysis indicates that 15 times latency reduction is possible for the preprocessing phase and 20 times latency reduction is possible for the query–response phase in the HXT scheme compared to the parallelized software implementation. Therefore, it is evident that developing a dedicated parallelized hardware accelerator for SSE is crucial for its practical deployment in real-life applications. A general implementation framework for SSE with sound cryptographic techniques, faster implementations with hardware acceleration are essential for SSE and cloud security. This SSE implementation provides a sound and secure design that can be adopted by concerned organizations, and other establishments to build a secure infrastructure for clients, consumers, and citizens of states.

8.7 Conclusions

The article is a summary of the research activities in the topic of Hardware Security performed in our country. We have focused on activities done in the SEAL Lab, Department of Computer Science and Engineering, IIT Kharagpur. The paper deals with several topics in hardware security, starting from hardware design of public-key algorithms, to Side-Channel Analysis in the form of power and micro-architectural attacks. PUFs were discussed as promising primitives for IoT security. Finally, we discussed about lightweight search techniques on encrypted databases, and the potential to develop hardware accelerators for such operations.

Acknowledgements The author would like to thank, in particular, his former and present Masters and Ph.D students at the SEAL Lab, IIT Kharagpur for the various collaborations and findings. In particular, he would like to thank Arnab Bag, Durba Chatterjee, Debapriya Basu Roy, Manaar Alam, Sarani Bhattacharya, Sayandeep Saha, Sikhar Patranabis, Urbi Chatterjee for several discussions during the writing of this article.

References

1. Kocher P, Jaffe J, Jon B Differential power analysis. Advances in cryptology CRYPTO99. Springer, pp 388–397 (1999)
2. Boneh D, Millo R, Lipton R (1997) On the importance of checking cryptographic protocols for faults. Advances in cryptology EUROCRYPT97. Springer, pp 37–51
3. Biham E, Shamir A (1997) Differential fault analysis of secret key cryptosystems. In: B.S.K. Jr. (ed) Advances in cryptology–CRYPTO 1997. Lecture Notes in Computer Science, vol 1294. Springer, pp 513–525
4. Hankerson D, Menezes AJ, Vanstone S (2006) Guide to elliptic curve cryptography. Springer Science & Business Media
5. Christopher, ST (2010) Tarnovsky hacks infineon's 'unhackable' chip, we prepare for false-advertising litigation. www.Engadget.com
6. Sparks ER (2007) A security assessment of trusted platform modules. Technical report, Department of Computer Science Dartmouth College. http://www.cs.dartmouth.edu/~pkilab/sparks/
7. Pappu RS, Ravikanth PS, Recht B, Taylor J, Gershenfeld N (2002) Physical one-way functions. Science 297:2026–2030
8. Tunstall M, Mukhopadhyay D, Ali S (2011) Differential fault analysis of the advanced encryption standard using a single fault. In: Information security theory and practice.security and privacy of mobile devices in wireless communication, Springer pp 224–233
9. Tupsamudre H, Bisht S, Mukhopadhyay D (2014) Destroying fault invariant with randomization. In: Cryptographic hardware and embedded systems–CHES 2014, Springer, pp 93–111
10. Saha S, Mukhopadhyay D, Dasgupta P (2018) ExpFault: an automated framework for exploitable fault characterization in block ciphers. IACR Trans Cryptogr Hardw Embed Syst 2018(2):242–276
11. Saha S, Jap D, Patranabis S, Mukhopadhyay D, Bhasin S, Dasgupta P (2018) Automatic characterization of exploitable faults: A machine learning approach. IEEE Trans Inf Forensics Secur 1 (to appear). https://doi.org/10.1109/TIFS.2018.2868245
12. Güneysu T, Paar C (2008) Ultra high performance ecc over nist primes on commercial fpgas. In: International workshop on cryptographic hardware and embedded systems, Springer, pp 62–78
13. Rebeiro C, Roy SS, Mukhopadhyay D (2012) Pushing the limits of high-speed gf (2 m) elliptic curve scalar multiplication on fpgas. In: International workshop on cryptographic hardware and embedded systems, Springer, pp 494–511
14. Roy SS, Rebeiro C, Mukhopadhyay D (2013) Theoretical modeling of elliptic curve scalar multiplier on lut-based fpgas for area and speed. IEEE Trans Very Large Scale Integr (VLSI) Syst 21(5):901–909
15. Roy DB, Mukhopadhyay D, Izumi M, Takahashi J (2014) Tile before multiplication: an efficient strategy to optimize DSP multiplier for accelerating prime field ecc for nist curves. In: Proceedings of the 51st annual design automation conference, ACM, pp 1–6
16. Roy DB, Das P, Mukhopadhyay D (2015) Ecc on your fingertips: a single instruction approach for lightweight ecc design in gf (p). In: International conference on selected areas in cryptography, Springer, pp 161–177
17. Becker GT (2015) The gap between promise and reality: on the insecurity of xor arbiter pufs. In: International workshop on cryptographic hardware and embedded systems, Springer pp 535–555
18. Sahoo DP, Mukhopadhyay D, Chakraborty RS, Nguyen PH (2018) A multiplexer-based arbiter puf composition with enhanced reliability and security. IEEE Trans Comput 67(3):403–417
19. Chatterjee U, Govindan V, Sadhukhan R, Mukhopadhyay D, Chakraborty RS, Mahata D, Prabhu MM (2018) Building puf based authentication and key exchange protocol for iot without explicit crps in verifier database. IEEE Trans Dependable Secur Comput 1 10.1109/TDSC.2018.2832201
20. Rebeiro C, Mukhopadhyay D (2008) High speed compact elliptic curve cryptoprocessor for fpga platforms. In: International conference on cryptology in India, Springer, pp 376–388

21. Rebeiro C, Mukhopadhyay D, Bhattacharya S (2014) Timing channels in cryptography: a micro-architectural perspective. Springer
22. Bernstein DJ (2005) Cache-timing attacks on aes. Technical report
23. Bhattacharya S, Mukhopadhyay D (2015) Who watches the watchmen?: utilizing performance monitors for compromising keys of RSA on intel platforms. In: CHES. Lecture Notes in Computer Science, vol 9293. Springer, pp 248–266
24. Bhattacharya S, Mukhopadhyay D (2018) Utilizing performance counters for compromising public key ciphers. ACM Trans Priv Secur 21(1) 5:1–5:31
25. Bhattacharya S, Mukhopadhyay D (2016) Curious case of rowhammer: flipping secret exponent bits using timing analysis. In: CHES. Lecture Notes in Computer Science, vol 9813. Springer, pp 602–624
26. Liu F, Lee RB (2014) Random fill cache architecture. In: Proceedings of the 47th annual IEEE/ACM international symposium on microarchitecture, IEEE Computer Society, pp 203–215
27. Martin R, Demme J, Sethumadhavan S (2012) Timewarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. ACM SIGARCH Comput Arch News 40(3):118–129
28. Alam M, Bhattacharya S, Mukhopadhyay D, Bhattacharya S (2017) Performance counters to rescue: a machine learning based safeguard against micro-architectural side-channel-attacks
29. Rebeiro C, Mukhopadhyay D, Takahashi J, Fukunaga T (2009) Cache timing attacks on clefia. In: International conference on cryptology in India, Springer, pp 104–118
30. Lai S, Patranabis S, Sakzad A, Liu J, Mukhopadhyay D, Steinfeld R, Sun S, Liu D (2018) Result pattern hiding searchable encryption for conjunctive queries. In: Proceedings of the 2018 ACM conference on computer and communications security (To Appear)
31. Bag A, Patranabis S, Tribhuvan L, Mukhopadhyay D (2018) POSTER: hardware acceleration for searchable encryption. In: Proceedings of the 2018 ACM conference on computer and communications security (To Appear)