Security *is* an Architectural Design Constraint

PRASANNA RAVI, Temasek Labs, Nanyang Technological University
ZAKARIA NAJM, Temasek Labs, Nanyang Technological University
SHIVAM BHASIN, Temasek Labs, Nanyang Technological University
MUSTAFA KAIRALLAH, School of Physical and Mathematical Sciences, Nanyang Technological University
SOURAV SEN GUPTA, School of Computer Science and Engineering, Nanyang Technological University
ANUPAM CHATTOPADHYAY, School of Computer Science and Engineering, Nanyang Technological University

In state-of-the-art design paradigm, power, space and time efficiency are considered the primary design constraints. This approach adversely impacts the security of the device, which is often adopted as a countermeasure after the vulnerability is identified. In this position paper, we argue that across all levels of an architecture design, security stands in direct contrast with other performance objectives, and therefore, needs to be treated as a design constraint from the early design phase. We advocate a security-aware design flow starting from the choice of cryptographic primitives, protocols and system design.

Additional Key Words and Phrases: Digital systems, design constraints, security-efficiency trade-off, security-aware design

1 INTRODUCTION

Historically, design of computer systems have revolved around the notion of performance, primarily measured in terms of *time* and *space* efficiency. It was only with the advent of low-footprint portable devices during the 80's that *power* featured as a potential measure of performance. Careful scrutiny and evaluation over a couple of decades confirmed that power efficiency complements and contradicts the notions of time-and-space efficiency in practical systems, and hence, it should be considered as a fundamental constraint in design. We have come a long way since then, and for over a decade, we have unequivocally considered *time*, *space* and *power* as the three primary constraints of architectural design. In this paper, we argue that it is time to consider the notion of *security* as the fourth axis in the space of architectural design, as it orthogonally complements the landscape of time, space and power.

We base our argument on evidences of the eternal conflict between security and performance in architectural design. Similar to our notion of performance spread over the three dimensions of time, space and power, we present the notion of *security* across three layers — cryptographic primitives, security protocols and security systems — as depicted in Table 1. It is evident from the literature of cryptanalytic results and security breaches that each of these three layers demonstrate a fundamental trade-off between performance and security. Quite often, architectural designs focussed on the fundamental principles of time-space-power efficiency introduce security vulnerabilities in cryptographic primitives, security protocols and security systems. Most critical vulnerabilities are generally noticed in the systems layer, spread across the range of hardware systems, software systems and hardware-software interfaces, where the time-space-power design constraints are considered to be of highest priority. This is where we advocate the inclusion of *security* as a fundamental architectural constraint to complete the design landscape.

Primitive Level	Mathematical Models	Pseudo-Random Generators, Functions, and Permutations
	Key-usage Paradigms	Symmetric Key Cryptography, Asymmetric Key Cryptography
	Cryptographic Modules	Block Ciphers, Stream Ciphers, Hash Functions, Signatures, etc.
	Cryptographic Modes	Encryption, Authentication, Authenticated Encryption, etc.
Protocol level	Transport Layer Security, Secure Sockets Layer, Secure Shell, IP Security, Wireless Security, etc.	
System Level	Software Abstraction, Hardware Abstraction, Software-Hardware Interface, Operating Systems, etc.	

Table 1. The layers of Security — Cryptographic Primitives, Security Protocols and Security Systems

Authors' addresses: Prasanna Ravi, Temasek Labs, Nanyang Technological University, Singapore, prasanna.ravi@ntu.edu.sg; Zakaria Najm, Temasek Labs, Nanyang Technological University, Singapore, zakaria.najm@ntu.edu.sg; Shivam Bhasin, Temasek Labs, Nanyang Technological University, Singapore, sbhasin@ntu.edu.sg; Mustafa Kairallah, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, mustafam001@ntu.edu.sg; Sourav Sen Gupta, School of Computer Science and Engineering, Nanyang Technological University, Singapore, sg.sourav@ntu.edu.sg; Anupam Chattopadhyay, School of Computer Science and Engineering, Nanyang Technological University, Singapore, anupam@ntu.edu.sg.

Security-Efficiency Trade-Off

The trade-off between security and efficiency is as old as the dawn of cryptography. Formal notion of information-theoretic security, as introduced by Shannon in 1950s, warrants the use of perfectly random one-time pads, which are absolutely useless in terms of practical efficiency. Cryptography practitioners, hence, introduced the notion of pseudo-random generators to approximate the desirable properties of one-time pads, and we followed the path of computational security. In a similar vein, we discarded the inefficient symmetric key-exchange mechanism of Merkle Puzzles that provided a quadratic computational gap between the users and the adversary to adopt the efficient asymmetric key-exchange mechanism like Diffie-Hellman, providing an exponential advantage to the users. Even today, most of the theoretical proofs in security rely on the random-oracle property of compression functions, whereas practical instantiations could only remain efficient till the construction of standard hash functions. It is evident that security and efficiency do not go hand-in-hand, and in this paper, we provide practical evidences to argue that security generally comes at the cost of efficiency. We will henceforth adopt a top-down approach in this paper, wherein we refer to practical instances where this security-efficiency trade-off can be observed at various layers of the stack — starting with the abstraction of cryptographic primitives, progressively moving down to the implementations.

Depending on the application and the use case in hand, a secure design always looks to achieve a multitude of different objectives like fast performance, low resource utilization, low power consumption and many more. In this paper, we will concentrate on the three fundamental efficiency parameters — time, space and power. In the quest for optimizing these resources, practical instantiations of secure algorithms often render themselves vulnerable. There are various categories of such vulnerabilities, some of which are as follows:

- Inherent security-efficiency trade-off is done when deciding the various parameters for a given secure algorithm.
- Efficient instantiations of a given secure algorithm might yield very good performance compared to a random or a generic instantiation of the same, but the same efficient instance might pave way for unforeseen vulnerabilities.
- Cryptographic primitives when implemented in a *standalone* mode may be secure, but quite often, an efficient encapsulation of the primitive into a broader class of security protocols might lead to vulnerabilities.
- Careless optimization technique implemented on a secure algorithm might lead to leakage of information.
- Generic security-agnostic performance enhancement approach developed for a specific platform might lead to creation of side channels, thus weakening the implementation of any secure algorithm on the same platform.
- Certain efficient implementation strategies providing time-efficiency open gates to side channel leakage.
- Optimizations employed by (semi-)automated tools over a given implementation of a secure algorithm (in most of the case, the source code) might discard inefficient features that ensured security in the first place.

We broadly classify the literature of security vulnerabilities introduced due to performance improvements into three categories of efficiency — time, space and power — affecting the three layers of security — primitives, protocols and systems. The cross-layer cross-category taxonomy in context of this paper is set as `XXX.YY`, where `XXX` denotes the affected security layer, and `YY` denotes the efficiency node, which causes the security loophole.. The two-way table of references, in line with this taxonomy, is presented as Table 2. Sections 2, 3 and 4 present in details the evidences of security-efficiency trade-off from the literature in a more systematic format (layer-wise) to support our argument.

		Time Efficiency	Space Efficiency	Power Efficiency
		TE	SE	PE
Primitive Level	PRI	[5, 20–22, 27, 47, 54, 56, 63, 66, 79, 81, 87, 88]	[10, 12–14, 19, 27, 44, 47, 50–53, 55, 61, 63, 65, 66, 68–70, 79]	[10, 12, 13, 19, 27, 44, 47, 50, 51, 53, 55, 61, 68]
Protocol Level	PRO	[6, 35, 84]	[36, 46, 46]	-
System Level	SYS	[2–4, 8, 9, 15, 17, 18, 24, 29, 31, 34, 38–40, 42, 43, 48, 49, 58, 60, 67, 71, 72, 76, 77, 80, 83, 89–91]	[2, 4, 8, 9, 15, 18, 24, 31, 38, 39, 43, 48, 49, 58, 59, 72, 77, 78, 80, 83, 85, 85, 89, 91]	[82]

Table 2. Literature of Security-Efficiency trade-off : Layers of Security vs Efficiency Considerations

2 PRIMITIVE LEVEL

The computational security notion governing the security of both private key and public key cryptographic primitives are very well understood. While the security of public key cryptographic primitives are derived through polynomial time reductions from provably hard mathematical problems, security of private key primitives are derived from constructions like Feistel structures and SPN (Substitution and Permutation) networks governed by well defined mathematical concepts like confusion and diffusion. Though there have been a number of reported attacks and vulnerabilities of these primitives in literature [5, 21, 87], none of them are catastrophic but merely point out to the existence of certain corner cases, weak instances and insecure algorithmic optimizations. We would like to focus on such instances in this section that especially argue our case of the conflict between security and efficiency at the primitive level. We separately analyse classical public key, post-quantum public key and symmetric key cryptographic primitives.

2.1 Public Key Cryptography

The traditional public key cryptographic primitives like RSA and ECC based cryptographic systems used in almost all secure communications derive their security guarantees from hard problems based in the field of number theory. While the security of RSA depends on fatorization of a product of two large prime numbers, ECC relies on the hardness of solving the discrete logarithm problem. Though the underlying hard problems of these schemes are rendered intractable by classical computers, a number of weaknesses and vulnerabilities are known to have been exploited leading to practical attacks on the RSA and ECC based cryptographic schemes. And following the argument of our paper, it is not surprising to know that many of those vulnerabilities stem from the presence of the cross-layer phenomenon between security and efficiency, which will be covered in the following discussion.

2.1.1 Exploiting Reduced Entropy in KeyPair Generation (Type `PRI.TE`*).* Keypair generation is a very important procedure in both RSA and ECC based cryptography and reuse of randomness is a common implementation strategy used to improve efficiency, but this techinque does not have a good track record in security as it has lead to a number of well known attacks.

Exploiting reuse of operating group and primes: Adrian et al.[5] reported the famous "LogJam" attack in 2015, an MITM (Man In The Middle) attack on TLS connections in which servers could be tricked into using "Export Grade" Diffie Hellman that operated over 512-bit groups. The main vulnerability stemmed from the usage of same 512 bit group across 8.4% of Alexa Top Million websites and the same 1024 bit group 3.4% of all HTTP servers, thus a massive precomputation step could be used to amortize the attack time over multiple entities using the same group. Heninger et al. [56] performed the then largest network survey of TLS and SSH servers in 2012 and reported vulnerabilities due to usage of keys with insufficient entropy and usage of same key in shared hosting conditions. Another similar vulnerabililty due to reuse of *ephemeral* keys in Elliptic Curve Digital Signature algorithm was reported by a hacker group named *FailOverflow* on Sony PlayStation 3.

Exploiting use of efficient prime generation algorithms: Švenda et al.[81] performed statistical analysis on a large number of public key and moduli used for RSA generated from a variety of cryptographic libraries and smart cards and observed that a given key could be classified into its correct key source with a very high accuracy of 85%, thus exposing anonymity of users. This is due to the existence of multiple efficient algorithms for prime generation like random sampling method, incremental search algorithm, rejection sampling, use of "Square" regions etc. which leave an observable signature for themselves allowing for easy detection. The same authors further discovered that the prime generation algorithm used by the cryptographic library *RSAlib* from *Infineon Technologies AG* only generated primes that were of the form

$$p = k * M + (65537^a \bmod M)$$

wherein the RSA prime p generated only depends on a and k and M is known. The primes of this form were shown to be easily factorizable and also were easy to be fingerprinted due to the abnormal decrease in entropy.

2.1.2 Efficient Parameter Instantiations (Type `PRI.TE`*).* Modular exponentiation used in RSA algorithms is very costly in terms of performance and resource utilitzation. Thus, use of efficient parameters to speed up implementations is very common. For example, use of a small secret key exponent for signatures will significantly speed up signature generation, but Wiener [88] showed that private key exponents satisfying the bound $d < N^{0.25}$ where d, N are the private key exponent and modulus respectively leads to a break of the RSA cryptosystem. by Boneh et al. [21] to $d < N^{0.292}$. Similarly, usage of a small private key exponent has been shown to be exploited by a number of attacks like Hastad Broadcast attack [54], partial key exposure attack [22] using variants of the Coppersmith's theorem [].

2.1.3 Efficient Techniques for Modular Exponentiation (Type `PRI.TE`*).* The *Chinese Remainder Theorem* (CRT) is a well known efficient technique to perform modular exponentiation which computes over primes half the size as that of the original modulus leading to a speed-up upto a factor of four. But Boneh et al. [20] showed that a single fault injected during computation using one of the prime factors in a CRT optimized RSA signature generation procedure results in trivial retrieval of the key from the faulty signature. A recent report by Weimer [87] showed that this simple fault classical attack still poses a threat to real world systems using TLS with RSA signature schemes. The countermeasure against the fault attack leads to significant decrease in performance as it requires an additional signature verification and hence is not widely deployed.

2.2 Post Quantum Public Key Cryptography

The cross-layer phenomenon not only is observable in classical cryptography, but also extends its presence into post quantum cryptographic primitives. The cryptographic community is actively working towards standardization of quantum resistant public key cryptographic primitives, better known as "Post-Quantum" cryptography. There have been several proposals for post quantum cryptography from varied fields of mathematics among which lattice based cryptography and code based cryptography seem to be the more promising proposals that provide both quantum resistance guarantees along with practical efficiency comparable on a scale with traditional public key cryptography.

2.2.1 Lattice based cryptography (Type `PRI.TE` *&* `PRI.SE`*).* Lattice based cryptography, in its infancy was considered to be near impractical due to the schemes suffering from asymptotically large key sizes and operation counts ($\Sigma(n^2 log(n))$) where n is the security parameter. But, the security of these schemes were based on hard problems on general lattices which were considered to be $NP - Hard$ in the worst case, thus offering very good security guarantees. A lot of research then was focussed on increasing the efficiency of lattice based cryptographic schemes, with the main direction being development of schemes with hardness on algebraically structured ideal lattices [63, 66], yielding asymptotic efficiency in both space and time, with reduced key sizes and computation time ($O(nlog(n))$), since arithmetic could be done over polynomials in rings as opposed to matrix vector arithmetic in the case of general lattices. This triggered a large body of work towards efficient implementation of lattice based cryptographic primitives on a range of devices from the smallest 8-bit AVR microcontrollers [62, 75] to reconfigurable hardware [57, 74]. But, the caveat present here is that the same hard problems over the structured ideal lattices which determines the security guarantees of these efficient schemes are not known to be as hard as that on general lattices. Even with extensive cryptanalytic efforts on these structured variants [26, 30], there are not any known weakness still known that could be exploited from their algebraic structure. With many of the efficient lattice based cryptographic schemes basing their security over hard problems on algebraically structured lattices [7, 23], cryptanalysis of lattice based cryptographic schemes will be intensely scrutinized over the coming years.

2.2.2 Code based cryptography (Type `PRI.TE` *&* `PRI.SE`*).* One can also observe very similar trends in code based cryptography where there is a dilemma in a choice between structured but efficient instantiations as opposed to unstructured but inefficient instantiations of code based cryptographic schemes. The first code based cryptographic scheme, the McEliece encryption scheme [65] was proposed using binary Goppa codes, but these schemes suffered from large sizes of public keys along with complex decoding procedures. A large body of work concentrated on development of efficient but secure choices of algebraically structured linear codes like Reed-Solomon [14], Reed-Muller codes [79], quasi-cyclic and quasi-dyadic codes [69] and many more. But, most of them are known to be broken with only the initial proposal of the Binary-Goppa codes [65] and the QC-MDPC codes [70] still considered to be secure. According to the state of the art, the QC-MDPC code based schemes over very compact keys ($1 - 2KB$) while at the same time being very efficient, but have a certain error probability associated with their decryption procedures, which was shown to be exploitable through the GJS reaction attack reported in [52], provided the same key is used across many number of encryptions. But, the relateively inefficient binary Goppa code variant of the McEliece encryption scheme still stands unscathed even with about close to 40 years of cryptanalysis efforts.

2.3 Symmetric Key Cryptography

2.3.1 Security of Private-Key Primitives (Type `PRI.SE` *&* `PRI.TE` *&* `PRI.PE`*).* The security of all symmetric key cryptographic primitives are directly related to the size of the shared secret, which is commonly indicated by the bit security level. A bit security of n bits indicates that a black box attacker has to perform at the most 2^n operations to retrieve the secret key. The bit security level is determined based on the best known attacks against the symmetric key primitive and thus need not be equal to the bit size of the secret. Moreover, due to the sustainable decrease in the cost of computational power, recommended security levels for various cryptographic applications are regularly increased, with the most recent instance being the declaration of any security level below 112 bits to be insecure according to NIST [11], thus phasing out the use of PRESENT-80 [19] and LED-64 [51] light weight block ciphers. Thus, upgrading the bit security level of any symmetric key primitive would indicate increasing the bit size of the key, implying larger storage, more operations on the key and ultimately a larger resource footprint.

The area of lightweight cryptography has attracted a lot of attention which has spurred the development of many light weight cryptographic designs like efficient block ciphers (PRESENT, LED, SIMON/SPECK [12], SKINNY [13], GIFT [10]), stream ciphers (Grain [55], Plantlet [68], Fruit [44], Lizard [53]) and Hash Functions (PHOTON [50]). The main reason can be attributed to the emergence of embedded device technologies like Bluetooth, Internet-of-Things (IoT), Wireless Sensor Networks (WSNs), Wearable Devices etc. which primarily operate on low power over computationally constrained platforms. While most of these ciphers achieve competitive bit-security levels, they build upon less secure and more efficient building blocks leading to low resource consumption, but require a higher number of iteration rounds which adds up to processing time. Besides, since this field is relatively new, the security gap between these lightweight primitives and their old trusted counterparts (AES, SHA-2 [41] , SHA-3 [37]) has not been extensively studied, thus leading to restrain from using these lightweigtht designs from use in high security and sensitive applications.

2.3.2 Post-Quantum Security of Private-Key Primitives (Type `PRI.SE` *&* `PRI.TE` *&* `PRI.PE`*).* Unlike public-key primitives, there are no known quantum attacks on private-key primitives except for Grover's Search Algorithm [47], which can speed up brute-force search attack from 2^n to $2^{n/2}$. Hence, post-quantum private-key primitives have to be at least twice as large as their classical counterparts in order to achieve the same security level, which, again, shows the trade-off between efficiency and security [27].

2.3.3 Recent Case Study: Hardware Evaluation of CAESAR Candidates (Type `PRI.SE` *&* `PRI.TE` *&* `PRI.PE`*).* The CAESAR competition [1] was announced in 2013, to allow the academic community to choose a set of authenticated encryption algorithms to be studied. Over 5 years, more than 50 submissions have been intensively studied, evaluating their security, software performance and hardware efficiency. In March, 2018, the CAESAR competition was concluded by selecting 7 final proposals, divided into three use cases, as follows:

(1) Lightweight applications (resource constrained environments): ACORN and Ascon.
(2) High-performance applications: AEGIS, MORUS and OCB.
(3) Defense in depth: COLM and Deoxys-II.

In [61], the authors have studied the hardware performance, area and efficiency of all the third round candidates of the competition, by implementing them for ASIC. Their results showed that , when comparing ciphers designed for use cases segregated as lightweight and defense-in-depth applications, there is a clearly observable 10x gap in the throughput/area efficiency, where the lightweight candidates are significantly both faster and smaller than there defense-in-depth counterparts. Thus, all lightweight cryptographic designs clearly demonstrate instances of conflict between security and all types of efficiencies like Space efficiency (SE) through small designs, time efficiency (TE) through high throughput rates and power efficiency (PE) through reduced power consumption.

3 PROTOCOL LEVEL

In almost all real world systems, crytograhic primitives are not implemented in a *standalone* mode, but are encapsulated in a larger cryptographic protocol along with other cryptographic primitives to achieve different security objectives. The TCP/IP (Transmission Control Protocol/Internet Protocol) stack is one of the most used communication protocols used in most of the computer networks around the world. It has a modular architecture with multiple layers, with each layer secured with different cryptographic protocols that are required to interact with each other to provide end-to-end security. Incorporating such sucurity measures at each layer is considered costly sometimes, but there are several trivial attacks like Packet Sniffing, Sppofing, Cache Poisoning, Proxy routing table updates, DoS style of attacks and many more that are possible if all the layers are not properly secured. But, there have been multiple other instances where application of certain optimization techniqeus have compromised the security of even a provably secure protocol, with the Transport Layer Security (TLS) protocol being the main focus of this section.

3.1 Data compression techniques used in TLS protocol

Transport Layer Security (TLS) (Previously known as Secure Sockets Layer (SSL)) is one of the most widely used cryptographic application in the world, which mainly provides security to the transport-layer of the TCP/IP stack. Data compression techniques were widely being utilized to decrease network traffic congestion, but this compression mechanism leaked information about the internal state of the data. This has been known to be exploited by a number of vulnerabilities like BREACH [46], CRIME [36] and TIME [16] attacks.

3.1.1 CRIME Attack (Type `PRO.SE`*).* Both HTTP requests from the client and responses from the servers in cleartext are typically compressed by the TLS protocol using the DEFLATE[1] compression technique before they are encrypted to be sent over the insecure channel. Juliano Rizzo and Thai Duong [36] reported *Compression Ratio Info-leak Made Easy* (CRIME), a side channel attack that can retrieve information about session tokens and cookies. The attacker maliciously injects information into the victim's HTTP request and observes the size of the encrypted request. By adaptively altering the injected information depending on the observed sizes of the encrypted requests, an attacker can easily deduce information regarding some secret tokens embedded in the HTTP request.

3.1.2 TIME Attack (Type `PRO.SE`*).* Following the CRIME attack, the major vendors deprecated the use of TLS compression technique at both the client and server sides which successfully thwarted the CRIME attack. Later, Tal Be'ery and Amichai Shulman reported *Timing Info-leak Made Easy* (TIME) attack [16], a variant of the CRIME attack but mainly targetting HTTP responses. The attacker carefully crafts additional information to be padded into the victim's HTTP requests and observes a larger RTT (Round Trip Time) for those manipulated requests in which the added information matches wih the internal data. Using the observable time difference due to compression, the attacker can retrieve internal information about the HTTP responses.

3.1.3 BREACH Attack (Type `PRO.SE`*).* Gluck et al. [46] reported a variant of the CRIME attack called the *Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext* (BREACH) attack, which targets the size of the HTTP compressed responses (instead of TLS compressed requests as in CRIME) to reveal secret information about secret tokens and cookies in the body of the response.

3.2 Attacks on the encryption mode used in TLS protocol

There is another class of attacks that specifically target the CBC (Cipher Block Chaining) mode of encryption used in the context of the TLS protocol. Block ciphers are usually used in different modes to encrypt large amounts of data, ECB (Electronic Code Book) mode, Counter mode and CBC (Cipher Block Chaining) mode to name a few. The CBC mode for block ciphers is known for its efficient properties like reuse of ciphertext as initialization vector during encryption and its ability to decrypt data in parallel. The CBC mode in a standalone configuration is secure, but has caused a lot of security concerns when used in the TLS protocol [6, 35, 84].

3.2.1 BEAST Attack (Type `PRO.TE`*).* TLS records are typically first authenticated using the HMAC construction, padded with deterministic data to align the data to the block size and then encrypted. Different error responses were invoked for the cases when the padding is correct but the HMAC was wrong or when the padding itself is wrong. The attacker tweaks the ciphertexts to evoke response regarding the correctness of the padding to reveal information about the plaintext. This attack which was first published by Vaudenay [84] was later shown to be practical by Duong et al. [35] in 2011, famously known as the BEAST attack. The BEAST attack was made possible due to a number of reasons, but one of the two main reasons were the differential error response on either incorrect padding or authentication and use of the last ciphertext of the previous packet as the IV of a new packet for want of time efficiency, with the attack very well aided by the structure of the CBC mode. The attack could be thwarted by evoking the same response for both incorrect padding and incorrecting authentication. But, in doing so, the sender has to recalculate both the padding and the MAC, even in cases when packet has been correctly authenticated, increasing the computation times upon failure.

[1]https://tools.ietf.org/html/rfc1950

3.2.2 *LUCKY13 Attack (Type* `PRO.TE`*).*

Even on evoking the same response from the server upon failure due to different reasons to avoid information leakage, the attacker can still learn about the number of padded bytes based on the time taken for authentication. A padding error evokes a faster error response but the attacker can observe a slower response upon correct padding but incorrect authentication. Here again, the timing leakage is caused due to want of efficiency to avoid calculating the MAC even upon noticing an incorrect padding. Thus, fixing this requires the server to calculate MAC for both correctly and incorrectly padded messages. But, there existed still existed a timing leakage due to the difference in times for authenticating the data which gave information about the number of padded bytes. This attack which is very similar to the BEAST attack but uses a timing oracle was proposed by Nadhem AlFardan and Kenny Paterson [6], which is famously known as the LUCKY13 attack.

The above attacks led to abandoning the use of CBC mode atleast in the context of TLS protocol as MAC-then-encrypt along with CBC has too many issues which could not be resolved very easily, while also looking like a source of many other hidden vulnerabilities.

4 SYSTEM LEVEL

The theoretically secure cryptographic primitives and the corresponding cryptographic protocols are ultimately required to be implemented on real world systems through which one can leverage their security guarantees. They are implemented in a wide array of real world systems, ranging from the smallest micro-controllers used in wireless sensor networks to the most powerful general purpose computers powering the data centres. System designers are always in pursuit of efficient yet secure implementations of cryptographic primitives and protocols as they are always considered to be adding a significant overhead in terms of efficiency to the application in hand. This endless pursuit of implementation efficiency has been constantly plagued with security issues as well. Thus, following the trend observed in the higher layers of the cryptographic stack, we observe that this cross layer phenomenon has made its presence felt in various aspects of system level security as well, which will be covered in this section. We would like to diversify the section into three parts namely

(1) Hardware Security
(2) Software Security
(3) Hardware/Software Interface Security

4.1 Hardware Security

Hardware security as a discipline encompasses multiple fields dealing with cryptographic engineering and security such as hardware trojans, physical attacks, protection of the IC supply chain both pre-silicon and post-silicon, development of hardware root of trust and security enhanced hardware infrastructure. But following the line of work from the previous sections, we will focus on hardware security challenges from the stand point of efficient implementations of cryptographic primitives and protocols. Side channel attacks (especially power analysis) and fault attacks usually pose as a major threat towards secure cryptographic designs from the standpoint of hardware security, which will be the main focus of this section.

4.1.1 *Generic Side channel protection approaches.*

Side-channel countermeasures against power analysis attacks have been developed on two different lines: leakage hiding and leakage randomisation.

Leakage Hiding Countermeasures

Leakage hiding aims at data independent processing which removes any side-channel basis. Dual-rail precharge logic (DPL) are a fair representative of this class of countermeasures [83]. DPL leads to over two times overhead both in area and time/performance. It suffers from two security vulnerabilities i.e. early propogation effect and routing imbalance [31]. Fixing any of these vulnerabilities leads to more elaborate design leading to area/performance overheads [18, 38] and thus compromising security with efficiency (Type `SYS.TE` and `SYS.SE`). Similarly, there are also known leakage countermeasures that work in the time domain that work by randomizing the occurence of the sensitive operations. But, Clavier *et al.* in [28] showed that desyncrhonizing the sensitive operations within a time window of t will only lead to a linear increase in the number of execution traces required by the attacker to perform the attack, while increasing t clearly hampers performance of the design (Type `SYS.TE`).

One of the most fundamental requirements for a side channel resistant design is to run in constant time so as to not leak information about data through side channels. But, constant time implementations are usually slower and time consuming to implement. For instance, the variable time operation of the base field operations in Wolfssl or Openssl implementations were exploited through timing attacks reported in [3]. Though not all timing variations are directly related to the secret, which might seem to mitigate known side channel attacks, they have also been shown to open the door to new attacks [34]. These instances can be classified under the Type SYS.TE.

Leakage Randomisation Countermeasures

The alternate line of countermeasures is built upon leakage masking which is used to randomise the leakage of sensitive computations. We will consider private circuits [59], which form the basis of all masked implementations which assume an attacker with very strong probing capabilities. A t-order private circuit requires the attacker to probe $t + 1$ signals to get intelligible inormation on $1 - bit$. Since each bit is represented by $t - bits$ of the masked implementation, the area/performance overheads are evident. A practical study on implementation aspects of private circuits on FPGA was reported in [78] which reported a very large overhead of about 38× in the number of slices and 9× in performance as compared to the unprotected design. They also demonstrated that CAD tools in the bid to decrease CLB utilization and increase performance, optimize away the security measures employed in the private circuit design and thus require to provide explicit constraints in order to prevent insecure optimizations. Thus, these instances can be classified under the Type SYS.SE.

Similarly, building efficient algorithmic level masking countermeasures for asymmetric key cryptographic primitives based on ECC and RSA also remains an elusive task. With a plethora of known attacks with different capabilities [40], development of an efficient yet secure countermeasure to thwart all known attacks and possible future attacks is a daunting task. For example, data re-randomisation countermeasure for secure ECC designs against the powerful single execution attacks yield almost a two times increase in computational time [29]. These instances fall under the Type SYS.TE.

4.1.2 Security Oblivious Power Management (Type SYS.PE*).* Power, energy and thermal management has become a very crucial characteristic in today's modern commodity hardware, right from the ubiquitous embedded systems that are battery powered to the power hungry enterprise level systems. For instance, Dynamic Voltage Frequency Scaling (DVFS) is one of the most used techniques for energy management, wherein power consumption is regulated based on runtime task demands, by controlling the two crucial factors that majorly contribute to power consumption of the device - voltage and frequency. Tang et al. [82] demonstrated a remote fault attack on the ARM Trustzone CPU possible due to a fundamental flaw in the security oblivious DVFS technique. Pervasive software access was provided to the hardware registers which were used to control the voltage and frequency parameter of the device, that allowed an attacker to inject faults into the computation through remote software commands. This instance clearly demonstrates the need to employ efficiency improvements in secure designs with extreme caution and also stresses the need for widespread security measures at every level of a secure design.

4.2 Software Security

For a long time, work on compiler optimizations have only focussed on ensuring functional coherence between the source code and its compiler optimized version [64, 73]. But there have been a number of works across literature that have revealed a very clear case of bumping into security errors where many a time compiler optimizations have lead to violating a security guarantee made by the original source code [33, 85, 90]. The formalization of this problem was first done by Silva *et al.* in [33] coining the term of *correctness-security* gap in compiler optimizations. This triggered a large body of work to study and diminish the effect of compiler optimizations on security [32, 90]. Silva *et al.* [33] point out to three types of vulnerabilities introduced by compiler optimizations.

(1) Information Leakage through Persistent State
(2) Code Elimination through Undefined Behaviour
(3) Side Channel Attacks

4.2.1 Information Leakage through Persistent State (Type SYS.TE*).* One of the most famous compiler optimizations that is known for its security flaws is the *dead store elimination* (DSE). A secret key residing in memory is usually overwritten with random data after use (or deleted), so as to avoid its persistence in memory. This scrubbed data is never read again by the program, so this is sensed by the compiler and thus the last instruction which accesses the memory location (i.e) the scrubbing instruction is optimized away thus leading to security issues. Though this issue has been known for quite sometime and has been claimed to be preventable through a variety of techniques [42, 60, 67], a recent work by Yang *et al.* [90] evaluated the known techniques used in various security projects and noticed that many of them are flawed. They propose a scrubbing-safe DSE optimization, but it still remains to be seen that DSE can truly be trusted to be secure. Another well known optimization is the *inline* function call, which eliminates overhead steps of explicitly calling a function. But it has the implication of merging of the stack frames of the caller and callee function. Thus, any secret variable used inside a function lives for a longer time as it now becomes a part of the callee function, which is a typical example of violating boundaries of trust-separated domains where a variable could trespass the boundary implemented by the programmer. Code motion is another widely adopted optimization technique, through which the compiler re-orders code by examining dependencies between the various instructions used. This might again lead to a situation of a persistence of a secret variable in memory for a longer duration that desired.

4.2.2 Code Elimination through Undefined Behaviour (Type SYS.SE*).* Wang *et al.* [85] point out to a curious case of software bugs which they term as *Optimization Unstable* codes. These are code segments that can invoke undefined behaviour by the program, for eg. divide by zero, referencing a null pointer, shifting an *n* bit integer by more than *n* places etc. These type of codes are deemed to function erratically and thus the compilers more often than not take advantage of these code segments and optimize them away under the assumption that these undefined behaviours do not exist. Thus, any sanity checks like checking for an integer overflow or a null pointer reference will always evaluate to false and will be considered to be dead code by the compiler to be optimized away. These optimizations can sometime result in vulnerabilities based on buffer overflow or memory allocation. Wang *et al.* [85] also make a crucial observation of a general trend in compilers becoming more and more aggressive with successive generations in doing away with codes with undefined behaviour thus rising security concerns about their optimization characteristics.

4.2.3 Information Leakage through Side Channel (Type SYS.TE*).* Compiler optimizations are also considered to be notorious in removing certain timing guarantees of the source code that were placed intentionally by the developer to ensure constant time implementations. Well known optimizations like strength reduction, Peephole optimizations, Common Subexpression Elimination etc. are techniques typically used by compilers for combination, simplification and replacement of certain parts of codes trying to achieve more performance and lesser resource utilization. But, usage of these optimizations more often than not, could lead to possible exploits through a number of side channel attack vectors. Time critical parts of the code are sometimes written in *inline* assembly to ensure that compilers do not perform any alteration leading to vulnerabilities. Similarly, there are countless instances littered across literature dealing with compiler optimizations that clearly demonstrate yet another clear case of trying to reach the ever elusive sweet-spot that remains hidden between the security and efficiency guarantees.

In complex applications like Internet of Things (IoT) and smart autonomous cars, developers are using many generic complex software stacks wherein most cases, the knowledge of the low level architechture behaviour is abstracted away. In such a scenario, implementation of cryptographic primitives encapsulated in several software layers can lead to unforeseen security vulnerabilities. Automated countermeasure insertion against side channel attacks at compilation time from a high level abstract language is a deep research topic. Though design using high level languages makes the code easier to write, read and verify for implementation bugs and flaws, most of the underlying levels are abstracted away. Both the underlying micro-architechture and the compilation can introduce hidden information leakage [17, 71, 76]. The more the number of abstracted levels, lesser is the control over the actual behaviour of implementation, thus might lead to unforeseen security bugs and vulnerabilities.

4.3 Hardware/Software Interface Security

Apart from hardening devices against attacks purely exploiting vulnerabilities either in hardware or software, it is also important to know that there is also considerable leakage present at the interface between hardware and software, which is more commonly known as the *microarchitectural* level. The trillion fold increase in computational power over the last sixty years [45] can be attributed to a number of microarchitectural optimization strategies such as Cache memories, Pipelining, Branch Prediction, Multi-threading, out of order execution, virtualization etc. But, these optimizations also brought along with them hidden vulnerabilities that have been shown to be exploited by a number of attacks, which together can be bracketed under the term of *microarchitectural* attacks.

Cache Memory hierarchy - A necessary evil

The ever increasing gap between processor and memory speeds, is greatly attributed to the bisection of the semiconductor industry into two parts- Microprocessor and Memory [25]. While the microprocessor industry laid emphasis on increasing the speed of the processor, capacity had been the main driver for the memory industry. Cache memories were introduced in the 1960s in order to bridge this fast growing gap between processor and memory. Staring with a single level cache, architectures evolved to have upto 3 levels of caches - with the caches closer to the processor smaller and faster compared to the ones farther. There is an observable cache sharing hierarchy, wherein all cache levels except the Last Level Cache are local to the processor. This resource sharing which leads to an observable resource contention at multiple cache levels that allows for visibility of activities of other co-located entities that contest for resources at the same level. This granularity in visibility also increases as one moves up from the Last Level Cache (LLC) to the first level cache which is the closest to the processor.

Caches were traditionally only used to store instruction and data (Using D-cache for data and I-Cache for instruction) to increase system performance. But, there are also other smaller caches that are local to a processor core. Translational look-aside buffer, (TLB) which store page translation addresses used during page mapping and Branch Target Buffer (BTB) which store the branching addresses of upcoming branch instructions with the help of the Branch Prediction Unit (BPU) are a few examples of the same. In addition to speeding up information access from memory, the cache access times almost always depend on either the data value or the address or both. This differential behaviour of cache memory access towards data has been exploited in a wide variety of attacks which together can be referred to as *Cache Timing Attacks.*

Achieving Parallelism through Resource Sharing (Type `SYS.SE` and `SYS.TE`)

Instrucion level parallelism is another key objective that had been the main focus of architecture designers to improve processor resource utilization and achieve execution of multiple instructions per clock cycle. Several architectural level design optimizations like symmetric multi-processing, hardware multi-threading, out of order execution and speculative execution were used to achieve the afore mentioned objective. This resource sharing resulted in observable resource contention at a very fine level at various execution units like ALUs, FPUs, memory controllers, system buses, interconnects etc. Thus, any entity like a parallel thread, process or a Virtual Machine will be able to observe the footprint of other similar co-located entities on contended resources at the same level. For eg. one can observe resource contention of execution units and L1 cache at a thread level or between processes or VMs running on the same core, while the resource contention between two entities located on different cores is only observable at lower levels like the Last Level Cache, system bus etc. Resource contention at multiple levels acrorss the processor hierarchy renders visibility of behaviour of other co-locatd entities mainly through timing-channels, which has lead to a number of microarchitectural attacks.

4.3.1 Reported Attacks. Gu et al. [43], in their survey of microarchitectural timing attacks broadly classify the same based on two axes - according to the *level of sharing* and the *degree of concurrency* required for the attack. While an attacker at the top level enjoys a very fine grained visibility of a co-located victim's behaviour, an attacker working at the bottom, at the bus level can only observe something close to the overall throughput variation. Similarly, an attacker at the thread level only requires to perform pre-emptive multitasking, while an attacker at the Last Level Cache requires true concurrency with the victim process to perform the attack.

4.3.2 Based on Attack styles. Exploitation of resource contention at caches have been achieved through different attack styles like PRIME+PROBE, FLUSH+RELOAD, EVICT+TIME, MELTDOWN and SPECTRE. These are majorly side channel attacks that typically rely on timing leakage coming from resource contention observable across various levels of the cache hierarchy. Refer to Tab.3 for the description of the various reported styles of microarchitectural timing attacks.

There are other types of attacks that use cache as a covert channel to perform Denial of Service attacks [86] that can saturate the caches with the attacker's own data leading to serious performance degradation for the victim. These style of attacks have also been known to be performed on other types of caches like TLB [58] and BPU [2].

Class of Attack	**Description**
PRIME+PROBE	The attacker first *primes* the cache by filling it with its own lines in one or more sets. Once the victim finishes its execution, the attacker *probes* the previously loaded lines to observe any observable timing difference in the memory accesses.
FLUSH+RELOAD	An exact inverse of the PRIME+PROBE attack, wherein the attacker flushes the cache indexed by virtual addresses and lets the victim to execute. Once the victim has finished execution, the attacker reloads the same lines to check if the victim has accessed any of the same lines.
EVICT+TIME	A similar approach as that of the PRIME+PROBE attack, but first lets the victim run to observe the average run time. The attacker then evicts certain lines of interest and lets the victim run again to observe timing differences based on which some inference can be made on the victim's internal state.
MELTDOWN	Relies on execution of so called transient instructions (instructions which follow after a branch instruction or an exception) that are not meant to be executed by the victim. The Out of order execution technique used to increase time efficiency is exploited in this kind of attack, whose activity can be observed in the shared caches across various levels. MELTDOWN results in a privilege escalation vulnerability specific to Intel processors through execution of instructions after an instruction trap.
SPECTRE	These attacks exploit the speculative execution technique used to predict direction of branching instructions, thus relying on execution of transient instructions. Though the results of these transient instructions are thrown away, their footprints are not erased from the shared caches across various levels. These attacks which can result read arbitrary memory from victim's process, apply to Intel, AMD and ARM processors.

Table 3. Different styles of microarchitectural timing attacks exploiting resource contention at the shared cache memories

4.3.3 *Based on level of sharing:* (Type `SYS.SE` and `SYS.TE`)

Given the visibility of co-located entities rendered possible by resource sharing, attacks have been reported over a multiple of levels in the processor hierarchy. They are

(1) *Thread* Shared State
(2) *Core* Shared State
(3) *Package* Shared State
(4) *System* Shared State

An attacker present at the thread level usually observes contention at thread shared resources like ALUs [4], BTB [2], FPU [8], BPU [39], return stack buffers [24] etc. Attacks at the *Core* shared state [9, 15, 72] typically target activity in the L1 and L2 level caches due to contention among different threads and processes running on the same core. Attacks at the *Package* shared state [48, 49, 91] target activity in the Last Level Cache which is typically shared by multiple processors on the same core. These attacks typically have to work with lesser granularity in observing the victim's behaviour and require more concurrency with the victim entity residing on the other core. Contention on system level resources like System buses, processor interconnects and system interfaces like PCIExpress have also been shown to be exploitable by a number of covert channel, side channel and DOS style attacks [77, 80, 89].

With the above cited literature on microarchitectural attacks across the entire processor hierarchy, we can clearly see that major architectural optimization techniques have been employed without foreseeing the possible security threats. These instances observable at the interface between hardware and software again stand as evidence of the eternal conflict between security and efficiency.

5 PROPOSAL FOR A SECURITY AWARE DESIGN FLOW

In Sections 2,3,4, we surveyed as many references as possible available in literature from academia and the industry to provide evidence of the ever existing trade-off between security and efficiency across multiple layers of the cryptography stack. We could see that optimization strategies somehow always opened gates to some unforseeable security vulnerability or protecting against powerful attacks becomes a very costly affair from a designer's perspective, who always targets high performance and low resource utilization. Since vulnerabilities can be introduced at any level, a security engineer's job to ensure security at each and every level becomes a paramount task.

No longer can security be considered as an afterthought for a digital system design. Incorporation of security to existing digital systems using an ad-hoc approach has only lead to a number of attacks as seen in previous sections [16, 35, 36, 46]. Also some of the optimization techniques like cache memory hirarchy, out of order execution, speculative execution, branch prediction etc. were introduced long before security was being seriously considered as being a threat to digital computer systems. With an ever increasing number of types of attacks and the attack surfaces, it is now supremely important to integrate the notion of security into each and every level of the design flow of a digital system.

We would like to propose a fully security aware design flow that would be useful for a security engineer who is required to incorporate all the required and necessary security measures and functionalities for any given application. Refer Fig.1 for our proposal for a security aware design flow, which tries to incorporate security at each and every step of the design flow for both hardware and software implementations.

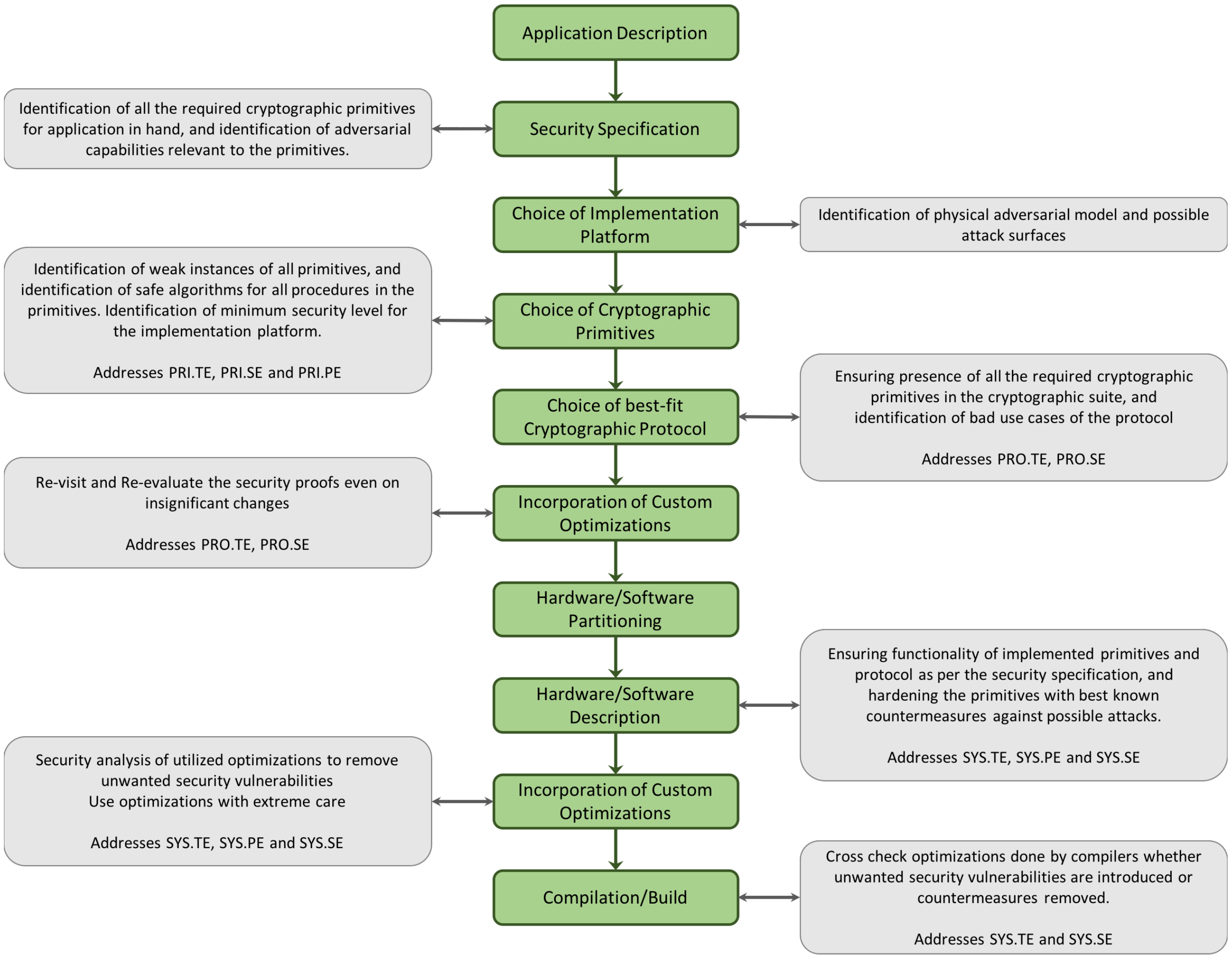
Application Description
Security Specification
Identification of all the required cryptographic primitives for application in hand, and identification of adversarial capabilities relevant to the primitives.
Choice of Implementation Platform
Identification of physical adversarial model and possible attack surfaces
Choice of Cryptographic Primitives
Identification of weak instances of all primitives, and identification of safe algorithms for all procedures in the primitives. Identification of minimum security level for the implementation platform.
Addresses PRI.TE, PRI.SE and PRI.PE
Choice of best-fit Cryptographic Protocol
Ensuring presence of all the required cryptographic primitives in the cryptographic suite, and identification of bad use cases of the protocol
Addresses PRO.TE, PRO.SE
Incorporation of Custom Optimizations
Re-visit and Re-evaluate the security proofs even on insignificant changes
Addresses PRO.TE, PRO.SE
Hardware/Software Partitioning
Hardware/Software Description
Ensuring functionality of implemented primitives and protocol as per the security specification, and hardening the primitives with best known countermeasures against possible attacks.
Addresses SYS.TE, SYS.PE and SYS.SE
Incorporation of Custom Optimizations
Security analysis of utilized optimizations to remove unwanted security vulnerabilities
Use optimizations with extreme care
Addresses SYS.TE, SYS.PE and SYS.SE
Compilation/Build
Cross check optimizations done by compilers whether unwanted security vulnerabilities are introduced or countermeasures removed.
Addresses SYS.TE and SYS.SE

Fig. 1. Our proposal for a Security Aware Design Flow

6 CONCLUSION

In this position paper, we have shown numerous examples about how security and efficiency stands in sharp contrast with each other. This is a fact that is not yet well understood in the design community, leading to regular and severe security breaches. The vulnerabilities presented in this paper shows security issues across multiple design layers and due to the pursuit to achieve different performance objectives, e.g., space, time and power. Consequently, we advocate a security-aware design flow that includes security as an architectural design constraint. This work calls for the development of an early design space exploration tool that includes security as a quantifiable metric. Further interesting directions could be to study the interplay between security wrappers in different design layers.

REFERENCES

[1] CAESAR Competition. https://competitions.cr.yp.to/caesar-submissions.html. Accessed: 2018-04-01.

[2] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In *Cryptographers Track at the RSA Conference*, pages 225–242. Springer, 2007.

[3] Onur Acıiçmez and Werner Schindler. A vulnerability in rsa implementations due to instruction cache analysis and its demonstration on openssl. In Tal Malkin, editor, *Topics in Cryptology – CT-RSA 2008*, pages 256–273. Springer Berlin Heidelberg, 2008.

[4] Onur Aciicmez and Jean-Pierre Seifert. Cheap hardware parallelism implies cheap security. In *Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on*, pages 80–91. IEEE, 2007.

[5] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM, 2015.

[6] Nadhem J Al Fardan and Kenneth G Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 526–540. IEEE, 2013.

[7] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-Quantum Key Exchange-A New Hope. In *USENIX Security Symposium*, pages 327–343, 2016.

[8] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 623–639. IEEE, 2015.

[9] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Fine grain cross-vm attacks on xen and vmware are possible! *IACR Cryptology ePrint Archive*, 2014:248, 2014.

[10] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift: a small present. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 321–345. Springer, 2017.

[11] Elaine Barker. Recommendation for key management part 1: General. *NIST special publication*, 2016.

[12] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The simon and speck lightweight block ciphers. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[13] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The skinny family of block ciphers and its low-latency variant mantis. In *Annual Cryptology Conference*, pages 123–153. Springer, 2016.

[14] Thierry P Berger and Pierre Loidreau. How to mask the structure of codes for a cryptographic use. *Designs, Codes and Cryptography*, 35(1):63–79, 2005.

[15] Daniel J Bernstein. Cache-timing attacks on aes. 2005.

[16] Tal BeâĂŽery and Amichai Shulman. A perfect crime? only time will tell. *Black Hat Europe*, 2013, 2013.

[17] Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and improvements of the DPA contest v4 implementation. In *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings*, pages 201–218, 2014.

[18] Shivam Bhasin, Sylvain Guilley, Florent Flament, Nidhal Selmane, and Jean-Luc Danger. Countering early evaluation: an approach towards robust dual-rail precharge logic. In *Proceedings of the 5th Workshop on Embedded Systems Security*, page 6. ACM, 2010.

[19] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 450–466. Springer, 2007.

[20] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *International conference on the theory and applications of cryptographic techniques*, pages 37–51. Springer, 1997.

[21] Dan Boneh and Glenn Durfee. Cryptanalysis of rsa with private key d less than n/sup 0.292. *IEEE transactions on Information Theory*, 46(4):1339–1349, 2000.

[22] Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on rsa given a small fraction of the private key bits. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 25–34. Springer, 1998.

[23] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Technical report, 2017.

[24] Yuriy Bulygin. Cpu side-channels vs. virtualization malware: The good, the bad, or the ugly. *Proceedings of the ToorCon*, 2008.

[25] Carlos Carvalho. The gap between processor and memory speeds. In *Proc. of IEEE International Conference on Control and Automation*, 2002.

[26] Hao Chen, Kristin Lauter, and Katherine E Stange. Attacks on search RLWE. https://www.microsoft.com/en-us/research/publication/attacks-on-search-rlwe/, 2015.

[27] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[28] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 252–263. Springer, 2000.

[29] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 292–302. Springer, 1999.

[30] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 559–585. Springer, 2016.

[31] Jean-Luc Danger, Sylvain Guilley, Shivam Bhasin, and Maxime Nassar. Overview of dual rail with precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors. In *Signals, Circuits and Systems (SCS), 2009 3rd International Conference on*, pages 1–8. IEEE, 2009.

[32] Chaoqiang Deng and Kedar S Namjoshi. Securing a compiler transformation. In *International Static Analysis Symposium*, pages 170–188. Springer, 2016.

[33] Vijay D'Silva, Mathias Payer, and Dawn Song. The correctness-security gap in compiler optimization. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 73–87. IEEE, 2015.

[34] Margaux Dugardin, Sylvain Guilley, Jean-Luc Danger, Zakaria Najm, and Olivier Rioul. Correlated extra-reductions defeat blinded regular exponentiation. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 3–22, 2016.

[35] T Duong and J Rizzo. Beast: Surprising crypto attack against https. *Blog, September*, 42:45–47, 2011.

[36] Thai Duong and Julianno Rizzo. The crime attack. https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide=id.g1d134dff_1_222.

[37] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

[38] Maik Ender, Alexander Wild, and Amir Moradi. Safedrp: Yet another way toward power-equalized designs in fpga. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 83–101. Springer, 2017.

[39] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Jump over aslr: Attacking branch predictors to bypass aslr. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–13. IEEE, 2016.

[40] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of secure ecc implementations: a survey on known side-channel attacks and countermeasures. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 76–87. IEEE, 2010.

[41] PUB FIPS. 180-4. *Secure hash standard (SHS), March*, 2012.

[42] FreeBSD. explicit_bzero - freebsd library functions manual. https://www.freebsd.org/cgi/man.cgi?query=explicit_bzero&sektion=3.

[43] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering*, pages 1–27, 2016.

[44] Vahid Amin Ghafari, Honggang Hu, and Ying Chen. Fruit-v2: ultra-lightweight stream cipher with shorter internal state. *IACR Cryptology ePrint Archive*, 2016:355, 2016.

[45] GIZMODO. The trillion fold increase in computing power, visualized. https://gizmodo.com/the-trillion-fold-increase-in-computing-power-visualiz-1706676799.

[46] Yoel Gluck, Neal Harris, and Angelo Prado. Breach: reviving the crime attack. *Unpublished manuscript*, 2013.

[47] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[48] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+ flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299. Springer, 2016.

[49] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *USENIX Security Symposium*, pages 897–912, 2015.

[50] Jian Guo, Thomas Peyrin, and Axel Poschmann. The photon family of lightweight hash functions. In *Annual Cryptology Conference*, pages 222–239. Springer, 2011.

[51] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The led block cipher. In *Cryptographic Hardware and Embedded Systems–CHES 2011: 13th International Workshop, Nara, Japan, September 28–October 1, 2011, Proceedings*, volume 6917, page 326. Springer, 2011.

[52] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on mdpc with cca security using decoding errors. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 789–815. Springer, 2016.

[53] Matthias Hamann, Matthias Krause, and Willi Meier. Lizard–a lightweight stream cipher for power-constrained devices. *IACR Transactions on Symmetric Cryptology*, 2017(1):45–79, 2017.

[54] Johan Hastad. Solving simultaneous modular equations of low degree. *siam Journal on Computing*, 17(2):336–341, 1988.

[55] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007.

[56] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, volume 8, page 1, 2012.

[57] James Howe, Ciara Moore, Máire O'Neill, Francesco Regazzoni, Tim Güneysu, and Kevin Beeden. Lattice-based encryption over standard lattices in hardware. In *Proceedings of the 53rd Annual Design Automation Conference*, page 162. ACM, 2016.

[58] Ralf Hund, Carsten Willems, and Thorsten Holz. Practical timing side channel attacks against kernel space aslr. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 191–205. IEEE, 2013.

[59] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[60] ISO/IEC. Iso/iec 9899:201x for c programming language. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf.

[61] Sachin Kumar, Jawad Haj-Yahya, Mustafa Khairallah, and Anupam Chattopadhyay. A Comprehensive Performance Analysis of Hardware Implementations of CAESAR Candidates. Technical report, 2018.

[62] Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede. Efficient Ring-LWE encryption on 8-bit AVR processors. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 663–682. Springer, 2015.

[63] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013.

[64] John McCarthy and James Painter. Correctness of a compiler for arithmetic expressions. *Mathematical aspects of computer science*, 1, 1967.

[65] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[66] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007.

[67] Microsoft. Securezeromemory function. https://msdn.microsoft.com/en-us/library/windows/desktop/aa366877(v=vs.85).aspx.

[68] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology*, 2016(2):52–79, 2017.

[69] Rafael Misoczki and Paulo SLM Barreto. Compact mceliece keys from goppa codes. In *International Workshop on Selected Areas in Cryptography*, pages 376–392. Springer, 2009.

[70] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo SLM Barreto. Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, pages 2069–2073. IEEE, 2013.

[71] Amir Moradi, Sylvain Guilley, and Annelie Heuser. Detecting hidden leakages. In *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, pages 324–342, 2014.

[72] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers' Track at the RSA Conference*, pages 1–20. Springer, 2006.

[73] James A Painter. Semantic correctness of a compiler for an algol-like language. Technical report, STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE, 1967.

[74] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 353–370. Springer, 2014.

[75] Thomas Pöppelmann, Tobias Oder, and Tim Güneysu. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 346–365, 2015.

[76] Pablo Rauzy, Sylvain Guilley, and Zakaria Najm. Formally proved security of assembly code against power analysis - A case study on balanced logic. *J. Cryptographic Engineering*, 6(3):201–216, 2016.

[77] Andre Richter, Christian Herber, Holm Rauchfuss, Thomas Wild, and Andreas Herkersdorf. Performance isolation exposure in virtualized platforms with pci passthrough i/o sharing. In *International Conference on Architecture of Computing Systems*, pages 171–182. Springer, 2014.

[78] Debapriya Basu Roy, Shivam Bhasin, Sylvain Guilley, Jean-Luc Danger, and Debdeep Mukhopadhyay. From theory to practice of private circuit: A cautionary note. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 296–303. IEEE, 2015.

[79] Vladimir Michilovich Sidelnikov. A public-key cryptosystem based on binary reed-muller codes. *Discrete Mathematics and Applications*, 4(3):191–208, 1994.

[80] WonJun Song, John Kim, Jae-Wook Lee, and Dennis Abts. Security vulnerability in processor-interconnect router design. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 358–368. ACM, 2014.

[81] Petr Švenda, Matúš Nemec, Peter Sekan, Rudolf Kvašňovskỳ, David Formánek, David Komárek, and Vashek Matyáš. The million-key question–investigating the origins of rsa public keys. In *25th USENIX Security Symposium. Proceedings*, 2016.

[82] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. Clkscrew: exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium*, 2017.

[83] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *Proceedings of the conference on Design, automation and test in Europe-Volume 1*, page 10246. IEEE Computer Society, 2004.

[84] Serge Vaudenay. Security flaws induced by cbc padding-applications to ssl, ipsec, wtls... In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 534–545. Springer, 2002.

[85] Xi Wang, Nickolai Zeldovich, M Frans Kaashoek, and Armando Solar-Lezama. Towards optimization-safe systems: Analyzing the impact of undefined behavior. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 260–275. ACM, 2013.

[86] Yao Wang and G Edward Suh. Efficient timing channel protection for on-chip networks. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 142–151. IEEE, 2012.

[87] Florian Weimer. Factoring rsa keys with tls perfect forward secrecy, 2015.

[88] Michael J Wiener. Cryptanalysis of short rsa secret exponents. *IEEE Transactions on Information theory*, 36(3):553–558, 1990.

[89] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security symposium*, pages 159–173, 2012.

[90] Zhaomo Yang, Brian Johannesmeyer, A Trier Olesen, Sorin Lerner, and Kirill Levchenko. Dead store elimination (still) considered harmful. In *26th USENIX Security Symposium. USENIX Association*, 2017.

[91] Yuval Yarom and Katrina Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732, 2014.