# Quantitative Analysis of Timing Channel Security in Cryptographic Hardware Design

Baolei Mao, Wei Hu, Alric Althoff, Janarbek Matai, Yu Tai, Dejun Mu, Timothy Sherwood, *Member, IEEE,* and Ryan Kastner, *Member, IEEE*

***Abstract*—Cryptographic cores are known to leak information about their private key due to runtime variations, and there are many well-known attacks that can exploit this timing channel. In this work, we study how information theoretic measures can quantify the amount of key leakage that can be exacted from runtime measurements. We develop and analyze twenty two RSA hardware designs – each with unique performance optimizations, timing channel mitigation techniques or discretization/randomization countermeasures. We demonstrate the effectiveness of information theoretic measures for quantifying timing leakage through correlation analysis of information theoretic measurements and attack results. Experimental results show that mutual information is a promising technique for quantifying timing leakage for RSA, AES and ECC ciphers, i.e., the mutual information correlates to being able to successfully guess the value of the private key. This is an important step towards a hardware security metric which allows designers to reason about security alongside traditional hardware design metrics like area, performance, and power.**

***Index Terms*—Hardware security, cryptographic function, timing channel, information flow, security metric.**

## I. Introduction

CRYPTOGRAPHIC algorithms are important functions for securing systems and information. These algorithms are constructed in a manner that makes them functionally difficult to break. For example, RSA (perhaps the most common asymmetric encryption algorithm) uses a trapdoor or one-way function related to prime factorization that makes it computationally difficult to attack when you do not know the private key. While these cryptographic algorithms are mathematically secure in a functional sense, their implementations may contain security vulnerabilities due to side channels. For example, researchers have demonstrated that it is easy to recover the secret key by looking at the amount of time the algorithm takes to execute (timing side channel) [1] or the amount of power it consumes while performing the computation (power side channel) [2].

There are a substantial number of demonstrated attacks on cryptographic systems using different types of side channels. Kocher was first to analyze the timing channels in several cryptographic algorithms [1]. Schindler et al. identified the timing channel in an RSA implemented using Chinese Remainder Theory [3]. The RSA implementation in the OpenSSL library was reported to leak timing information even in a noisy network environment [4]. More recently, the elliptic curve cryptography system in OpenSSL has also been disclosed to leak information from timing channel [5]. Yarom et al. successfully launched an attack on constant time RSA by exploiting cache timing channel [6]. The RSA blinding technique also failed to prevent timing leakage in a software RSA implementation [7].

Recently researchers proposed techniques to analyze hardware designs for timing channels or developed new or modified hardware languages that completely eliminated a specified timing side channel. Zhang et al. proposed a typing extension on top of Verilog that eliminated timing channels [8]. Li et al. developed a hardware description language that mitigated timing channels through careful finite state machine generation [9]. Oberg et al. [10] proposed a framework to detect timing channels in hardware designs including caches, cryptographic cores, and SoC systems.

These approaches generally take a *qualitative* "all or nothing" approach to timing channel security. In other words, they take (sometimes substantial) measures to completely eliminate the timing channel, or they simply detect that a side channel does or does not exist. And when there is a side channel, the techniques say nothing about the severity of that channel. Many times it is not possible or necessary to build a completely secure system (i.e., completely eliminate the side channel). This may be too costly in terms of performance, power, area, or design time; or it is not crucial to make the system 100% secure, e.g, the information being protected does not need the utmost levels of protection. Thus, cryptographic systems often allow for some amount of information leakage. For example, it may be ok to leak the position of the highest non-zero key bit if that will enable the design to execute substantially faster. Thus, we argue for more *quantitative* metrics that can assess the severity of the leakage, and enable the designer to trade-off the security benefits alongside traditional design metrics (e.g., area and performance).

Quantitative metrics can help designers more precisely mea-

Manuscript received April 30, 2017; revised August 25, 2017; accepted October 7, 2017. This work was supported in part by the NSF under Grant CNS-1527631, in part by the NSFC under Grant 61672433, in part by the Fundamental Research Funds for the Central Universities under grant 3102017OQD094, and in part by the Fundamental Research Fund of Shenzhen Science and Technology Innovation Committee under Grant 201703063000517. This paper was recommended by Associate Editor R., Sherief.

B. Mao, W. Hu, Y. Tai and D. Mu are with the School of Automation, Northwestern Polytechnical University, Xian, Shaanxi 710072, China (e-mail: maobaolei524@gmail.com; vinnie@mail.nwpu.edu.cn; taiyu@mail.nwpu.edu.cn; mudejun@nwpu.edu.cn).

A. Althoff, J. Matai and R. Kastner are with the Department of Computer Science and Engineering, University of California, San Diego, CA 92093 USA (e-mail: aalthoff@ucsd.edu; jmatai@cs.ucsd.edu; kastner@ucsd.edu).

T. Sherwood is with the Department of Computer Science, University of California, Santa Barbara, CA 93106 USA (e-mail: sherwood@cs.ucsb.edu).

sure the security of a design. They can answer design questions such as "is one implementation of a cryptographic function more secure than another?", and "how effective is a particular security mitigation technique?". These provide insights for making important design decisions. For example, A pipelined version of a design leaks 20% less timing information or adding some security mitigation technique will cause 15% more area overhead while making the design 40% more secure. Deriving effective metrics allows security be integrated as an important decision variable for hardware design space exploration.

Information theoretic notions developed by Claude Shannon provide powerful tools for quantifying channel capacity of a transmission medium [11]. For example, Clark et al. used entropy to quantify the amount of information leakage from "while" language [12]. Zhang et al. revealed information leakage of AES cryptographic implementation from cache architectures using mutual information analysis [13]. Köpf et al. studied the amount of information leakage versus performance and established boundaries on the amount of information flow [14].

This paper investigates how information theoretic metrics can be used to quantify timing side channels in cryptographic core implementations. Specifically, this paper makes the following contributions:

- Proposing metrics that enable designers to reason about the security of their hardware design with respect to timing side channels;
- Demonstrating how information theoretic methods such as *entropy* and *mutual information* quantify timing leakage across different cryptographic hardware architectures such as RSA, ECC and AES;
- Presenting experimental results that reveal the effects of synthesis optimization, mitigation techniques, countermeasures (discretization and randomization) on the reduction in RSA timing channel leakage using information theoretic measures and timing attacks;
- Validating and verifying the correlation between information theoretic measures and the ability to launch a timing attack, which allows information theoretic metrics to help designers to trade-off security alongside traditional design metrics such as area and performance.

The reminder of this paper is organized as follows. In Section II, we describe the threat model. In Section III, we cover timing attacks on RSA and our method for quantifying timing flow. Section IV discusses architecture optimization, mitigation techniques and countermeasures for creating different RSA architectures. Section V presents experimental results on RSA timing channel. Section VI presents timing channel evaluation about AES and ECC ciphers. We briefly review related work in Section VII and conclude in Section VIII.

## II. Threat Model

Cryptographic cores are usually integrated into a system on chip (SoC) to accelerate security functions. Our threat model assumes the attacker can measure encryption time, e.g., an untrusted IP core or malicious software/firmware in the SoC could monitor or use the cryptographic cores in order to recover the secret key through runtime measurements. Our threat model applies to any hardware components where the attacker can extract secret information through timing information. We use RSA as an example throughout the paper, but the idea can extend to other components where the pertinent information is a function of the computation time.

We assume the attacker knows the algorithmic details of the cryptographic core of hardware IP under attack. We further assume that the attacker has control over some inputs, (e.g., the plaintext) and can observe certain outputs, e.g., the ciphertext and "ready_out", "completed" or equivalent signal noting the end of the computation. Essentially we assume that the attacker has the ability to somehow measure the encryption time.

Our experiments are performed using different RSA architectures running on an FPGA. We add control logic around these RSA cores to provide input data, and determine the number of clock cycles required for each encryption operation. Our threat model can be extended beyond this experimental setup. It is applicable to any arbitrary hardware design such as AES or ECC that leaks information through a timing channel.

## III. Background

### A. Information Theoretic Metrics

***Entropy*** measures the uncertainty of a variable. Using $p(x)$ to denote the *probability mass function (pmf)* of random variable $X$, the *Shannon Entropy* is defined as

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{1}$$

***Mutual information*** quantifies the reduced uncertainty of a variable $X$ given the ability to observe another variable $Y$. It is a measurement of how much information variable $Y$ contains about variable $X$. The mutual information between $X$ and $Y$ is defined as

$$\begin{aligned} I(X;Y) &= H(X) + H(Y) - H(X,Y),\ x \in X, y \in Y \\ &= H(X) - H(X|Y) \end{aligned} \tag{2}$$

where $H(X|Y)$ is the *conditional entropy* of $X$ given $Y$.

### B. Timing Channel in RSA

RSA is a public key cipher that maintains a key pair for encryption and decryption. Given a public key $e$, secret key $d$, modulus $n$, and plain text $m$, the cipher text $c$ is encrypted and decrypted as follows:

$$\begin{aligned} c &= m^e \ mod\ n \\ m &= c^d \ mod\ n \end{aligned} \tag{3}$$

Modular exponentiation is the basic operation of RSA. Algorithms 1 and 2 illustrate how modular exponentiation is calculated through repeated square-and-multiply from right to left (*R-2-L*) and from left to right (*L-2-R*), respectively.

Both algorithms perform modular multiplication when the key bit under consideration has a binary value of `1` (corresponding to *Lines 4 - 5* in Algorithm 1 and *Lines 3 - 4* in Algorithm 2). By comparison, a simple assignment is performed when the current key bit has a value `0` (*Lines 6 -*

*7* in Algorithm 1 and *Lines 5 - 6* in Algorithm 2). Thus, the key bit values cause a timing variation in a sequential implementation. Furthermore, the messages (inputs) to the modular multiplication lead to variations in modular exponentiation time. These runtime differences create a timing channel that an attacker can use to ascertain information about the key.

**Algorithm 1** Modular exponentiation $c^d \ mod \ n$ calculated using square-and-multiply from right to left (*R-2-L*)

```
1: m[0] := 1
2: s[0] := c
3: for i := 0 to w − 1 do
4:    if d[i] == 1 then
5:       m[i + 1] := m[i] * s[i] mod n
6:    else
7:       m[i + 1] := m[i]
8:    end if
9:    s[i + 1] := s[i] * s[i] mod n
10: end for
11: Return m[w]
```

**Algorithm 2** Modular exponentiation $c^d \ mod \ n$ calculated using square-and-multiply from left to right (*L-2-R*)

```
1: s[w] := 1
2: for i := w − 1 to 0 do
3:    if d[i] == 1 then
4:       m[i] := s[i + 1] * c mod n
5:    else
6:       m[i] := s[i + 1]
7:    end if
8:    s[i] := m[i] * m[i] mod n
9: end for
10: Return m[0]
```

To demonstrate the existence of a timing channel, we create hardware designs for both the *L-2-R* and *R-2-L* algorithms and measure their execution time for different messages and keys. Figure 1 shows the runtime measurements in terms of the number of clock cycles. The two graphs at the top of Figure 1 show the runtime distributions for different messages using a single key (0x00971f1fbd396d4a4557ca2efa360475). The graphs at the bottom of Figure 1 present the runtime distributions for different keys using the same message (0x134001e5135cb206920021e5135cb206). The key and the message affect the computation of the modular exponentiation in different manners; the key dictates the control of each iteration and the message contributes to time to perform the modular multiply and modular square calculations. From Figure 1, we can see that the runtimes for different messages range from 93399 to 94630 clock cycles; *R-2-L* and *L-2-R* have the mean values of 94242 and 93750 clock cycles, respectively. The runtimes for different keys range from 78801 to 105983 clock cycles with the frequency below 8; *R-2-L* and *L-2-R* have the mean values of 93090 and 92598 clock cycles, respectively. This shows that both the message and key can cause variations in the runtime of the RSA algorithm, which makes it susceptible to a timing attack.

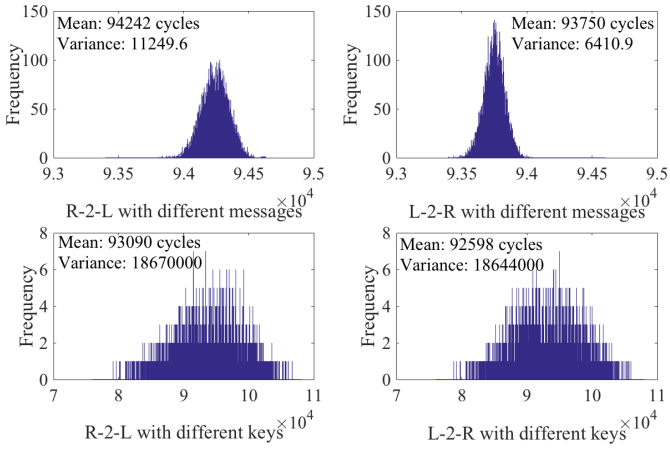


Fig. 1. RSA modular exponentiation time distribution for R-2-L and L-2-R algorithms. Modular exponentiation time in X axis is in clock cycles.

We will show how secret key can be recovered through simple yet effective statistical analysis of runtime measurements in Sections III-C and III-D.

### C. Kocher's Timing Attack

Kocher was the first to provide a comprehensive theoretical analysis of a timing attack using variance analysis [1]. The attack is based on the assumption that the runtimes for processing different key bits are independent, i.e., given a number of messages, the runtime observations for different key bits compose independent random variables.

Let $\boldsymbol{T}$ denote the vector that contains the total runtime observations for processing $N$ messages and $\boldsymbol{t}_i$ $(i = 0, 1, \cdots, w-1)$ denote the vector that contains the runtimes for the computation corresponding to the $i$-th key bit. Then we have

$$var(\boldsymbol{T}) = var(\sum_{i=0}^{w-1} \boldsymbol{t}_i) = \sum_{i=0}^{w-1} var(\boldsymbol{t}_i) \tag{4}$$

The attack makes guesses of the current key bit (assuming it could be either zero or one) and obtains the runtime vectors $\boldsymbol{t}_i^0$ and $\boldsymbol{t}_i^1$ through observation. For a correct guess $\boldsymbol{t}_i^c$ $(c \in \{0, 1\})$, the $var(\boldsymbol{T})$ will decrease by $var(\boldsymbol{t}_i^c)$ to $var(\boldsymbol{T} - \boldsymbol{t}_i^c)$. For the incorrect guess, $\boldsymbol{t}_i^{1-c}$ is independent from the correct runtime observations, and $var(\boldsymbol{T} - \boldsymbol{t}_i^{1-c})$ should theoretically increase $var(\boldsymbol{T})$ by $var(\boldsymbol{t}_i^{1-c})$. In a real attack, the runtime observation vectors are not perfectly independent from each other. However, a correct guess tends to decrease the variance by a larger amount than a wrong guess. The attack takes the guess value that results in a larger reduction in variance as the key bit.

### D. Sliding Window Timing Attack

Kocher's timing attack method is based upon the assumption that the runtime distribution for individual key bits is independent. It is possible to guess more than one bit per guess iteration as long as this independence condition still holds. We extend Kocher's timing attack to a *Sliding Window* method. In this method, we consider $l$-bits per guess iteration, where

$l$ is the window size. This yields to $2^l$ potential guesses at each guess iteration. A guess that gets all $l$ bits correct should result in the most significant decrease in variance while there should be a smaller decrease in the variance as the number of incorrectly guessed key bits increases.

In this following, we will introduce the *Non-overlapping Sliding Window* method (with window sizes of 1, 2, and 3) and *Overlapping Sliding Window* method (with a window size of 3) to perform timing attack and analysis. We show that small window sizes increase the accuracy over Kocher's method while running in a reasonable amount of time.

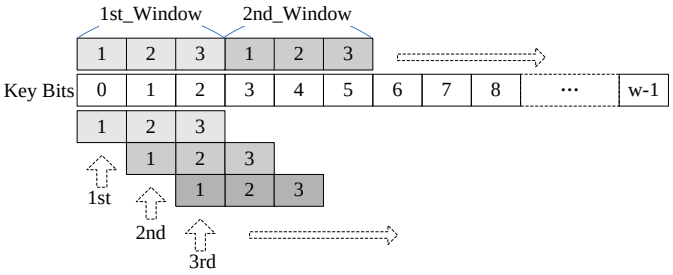


Fig. 2. The sliding window methods using 3-bit-window. The upper part shows the *Non-overlapping Sliding Window* method and the bottom part shows the *Overlapping Sliding Window* method.

The upper part of Figure 2 shows the *Non-overlapping Sliding Window* method using a 3-*bit-window*. The first window considers bits 0-*to*-2 of the key, resulting in 8 guesses: *000*, *001*, $\cdots$, *111*. The correct guess among the eight leads to the largest decrease in variance while the completely wrong guess should have the smallest decrease in variance. The decision on the current key bits can be made by observing and comparing the decrease in variance. Then we proceed to the next window that covers the next three bits of the key.

The bottom part of Figure 2 illustrates the *Overlapping Sliding Window* attack. Here, we determine the guess for the 3-*bit-window* as before, but the successive windows provide multiple guess results for the same bit. Thus, each key bit has three (or more depending on the window size) guesses corresponding to the number of windows covering it. We use majority vote to decide the current key bit. Note that the beginning and ending key bits have fewer than three guesses as they are a boundary condition.

The sliding window methods can improve the attack success rate. However, this comes at the cost of additional attack time as the window size increases. This is because the attack time is linear to the number of guess iterations, which increases exponentially with the window size. We focus on how many guess iterations are required for guessing each key bit on average. In order to make a decision on each key bit, 1-*bit-window* needs two guess iterations. With a window size of $l$, the attack has to perform $2^l$ guess iterations for each window, each key bit requires $2^l/l$ iterations on average. If the attack time is one unit for Kocher's attack, the attack time for the *Non-overlapping Sliding Window* (window_size = $l$) method can be described as:

$$cost_{sliding} = 2^l \ / \ 2^1 \ / \ l = 2^{l-1}/l \tag{5}$$

and the cost for the *Overlapping Sliding Window* method is:

$$cost_{overlap} \approx l * 2^{l-1}/l = 2^{l-1} \tag{6}$$

It is worth noting that window size selection should satisfy the independence assumption, i.e., the distributions of processing time for each window should be independent. Equations (5) and (6) show that the attack complexity grows exponentially as the window size increases. Therefore, we limit the window size to satisfy that $l$ is significantly smaller than $w$, where $l$ is window size, and $w$ is key length.

### *E. Quantifying Timing Information Flow in RSA*

In this section, we characterize the timing information leakage and establish a quantitative analysis model for the RSA timing channel.

We use $T$ to denote the set of RSA encryption times under different keys ($K$) and messages ($C$). For a given key $k \in K$ and message $c \in C$, we have a specific runtime observation $t \in T$. Since the keys and messages are discrete variables, we use the *pmf* to describe the distribution of the RSA runtime measurements as Equation (7):

$$L(K, C) = p\{T = t \mid C, K\}, \ c \in C, k \in K \tag{7}$$

The *pmf* describes the timing characteristics of the RSA architecture. For example, Figure 1 shows that different RSA implementations can have variations in runtime distribution. We can quantify the timing channel capacity of an RSA architecture in terms of its entropy:

$$H(T) = -\sum_{k \in K, c \in C} p(T = t) \log p(T = t) \tag{8}$$

The entropy quantifies the average amount of information revealed by runtime measurements. It only takes runtime observations into account, ignoring the effect of the key on the runtime. Although entropy provides a coarse-grain way to evaluate the timing channel capacity, it is incapable of quantifying how much information is leaking from each key bit in RSA architecture.

We use mutual information to disclose this bit-level timing channel characteristics about key information leakage. Given a set of keys, there will be corresponding runtime observations for these keys. We employ mutual information to measure the amount of information revealed from runtimes about the key bits:

$$I(k_i; T) = H(k_i) + H(T) - H(k_i, T) \tag{9}$$

where $k_i$ denotes the value of the $i$-th key bit and $T$ is the observed runtime for the entire key.

This shows how much information from the $i$-th key bit leaks through the execution time or equivalently how much the execution time depends on the $i$-th bit of the key. It also reveals the timing characteristics of the RSA architecture with respect to individual key bits. Although we quantify timing channel in RSA hardware in much detail throughout the paper, mutual information analysis can also apply to other ciphers such as AES and ECC. We will present timing channel evaluation for these ciphers in the following sections.

## IV. RSA Hardware Architetures

In this section, we will create 22 different RSA architectures, divided into two categories. The first category is generated by using different synthesis optimizations, which will be discussed in more detail in Section IV-A. The second group employs different timing channel mitigation techniques or discretization/randomization countermeasures, which will be covered in Sections IV-B and IV-C.

### A. RSA Performance Optimization Architecture

Hardware optimizations can have a significant impact on timing. For example, architectures exploiting parallelism or pipelining require a smaller number of clock cycles than those that work in a sequential manner. We are interested in understanding if and how these different optimizations affect the amount of information leaked through runtime. In order to evaluate our method, we use high-level synthesis (HLS) to do different hardware optimizations. Optimizations are typically specified using *pragmas* that tell the HLS tool how to optimize particular regions of the code, e.g., *pipeline*, and *unroll*. In this work, we use the *Xilinx Vivado HLS* tool to generate five different RSA architectures using various optimization strategies (e.g., to generate hardware with different performance and area trade-offs).

L1: *ModExp* (*Outer_Loop*)

```
for(i=0 to w-1){
  #pipeline
  #unroll
  if(Key[i]==1)
    ModMult(…); // mod multiply
  ModMult(…);  // mod square
}
```

L2: *ModMult* (*Inner_Loop*)

```
for(i=0 to w-1){
  #pipeline
  #unroll
  if(…) {
      ⋮
  }
}
```

Fig. 3. The basic RSA algorithm implemented with two nested loops. The outer loop calculates modular exponentiation; the inner loop performs modular multiplication.

We implement Algorithm 1 in synthesizable C code, which consists of two nested loops as shown in Figure 3. The outer loop (*L1*) performs computations from *Lines 3-10* in Algorithm 1. The inner loop (*L2*) performs the modular multiply or modular square in *Line 5* or *Line 9*. While there are many pragmas for optimization in HLS, we focus on *pipeline* and *unroll* due to their importance and impact on final architecture of hardware. The *pipeline* pragma is used to pipeline the iterations of loops. The *unroll* pragma allows multiple iterations of the loop to be executed at the same time.

Using these two pragmas, we generate 16 different architectures using different HLS directives. However, only 5 of them are relatively unique, i.e., the others are similar to these five and not interesting enough to discuss. The *OPTIMIZATION* group of Table I summarizes these five designs. The dash symbol indicates no optimization is performed for that loop.

The *Sequential* design does not have any optimization; it is largely sequential. The second design pipelines the modular multiply (*L2*). The *Unroll* design unrolls the modular multiply loop. The *Pipeline&unroll-1* design partially unrolls and pipelines the modular multiply loop. The *Pipeline&unroll-2* design unrolls the modular multiply loop and pipelines the modular exponentiation loop.

### B. RSA Timing Mitigation Architecture

Mitigation techniques make changes to the algorithm itself in order to reduce timing-based leakage. These mitigation techniques typically fall to two categories, either making runtime measurements constant or random [4], [15]. Other possible defenses attempt to decouple runtime measurements from messages. These include performing dummy modular multiplication even when the key bit is zero [4], moving the modular square into the conditional branch statements [4], inserting additional reduction in the Montgomery algorithm even if unnecessary [15], and introducing some random numbers into RSA computation (i.e., RSA blinding) to make the runtime observation unrelated to the message [4]. In our analysis, we consider several RSA implementations with built-in mitigation techniques. These designs include the *Left-to-right-multiply-always* (*L-2-R-always*), *Power-ladder*, *Montgomery-multiplication* (*Montgomery*), *Parallel*, *Exponent-blind* and *Base-blind* designs.

The *L-2-R-always* algorithm inserts a dummy multiply in the else statement of the conditional branch. This reduces the key dependent delay difference and helps mask the timing feature that causes key leakage. The *Power-ladder* algorithm carefully re-designs the algorithmic flow. It moves the modular square operation into the conditional branch and always performs both modular multiply and square regardless of the current key bit. For these previous two architectures, the runtime of a modular multiplication operation is not constant; there is timing variation for different messages. The *Montgomery* algorithm uses a different modular multiplier (i.e., MontMult). The runtime of a modular multiplication operation using this new multiplier is entirely determined by the modulus. Although there is still a timing difference caused by the conditional branch in the algorithm flow, it eliminates the timing difference resulting from different messages. Thus, the variance of the runtimes is consistently zero for both key bit guesses, making it impossible to determine the correct key bit. The *Parallel* design uses two modular multipliers to implement modular multiply and modular square, respectively. There is no data dependence between the conditional branch statement and modular square, these two components run in parallel. Because modular multiply always costs equal or less time than modular square, modular square execution time masks conditional branch statement execution time. The *Parallel* architecture reduces timing variation by increasing parallelism but it does not eliminate the timing difference resulting from different messages and keys. The *Exponent-blind* algorithm introduces a random number to protect the private key, which is based on Fermat's theorem to guarantee the correctness of modular exponentiation. It decouples and fuzzes the correlation between the private key and the runtimes significantly. The *Base-blind* algorithm introduces a random number to mask the messages, which makes the runtime observation unrelated to the messages. The *MITIGATION* group of Table I summarizes the different designs with mitigation techniques used in our analysis.

TABLE I
DESIGN OVERHEADS FOR DIFFERENT RSA ARCHITECTURES.

| Designs | | Optimization Techniques L1 | Optimization Techniques L2 | Mitigation Techniques | Slice Regs. | LUT | Area (Cells) | Avg. Cycles |
|---|---|---|---|---|---|---|---|---|
| Optimization | Sequential | – | – | – | 1846 | 1400 | 4479 | 93066 |
| | Pipeline | – | pipeline | – | 1851 | 1689 | 4919 | 70167 |
| | Unroll | – | unroll | – | 4204 | 4799 | 11779 | 75869 |
| | Pipeline&unroll-1 | – | pipeline unroll | – | 4212 | 4768 | 11939 | 70988 |
| | Pipeline&unroll-2 | pipeline | unroll | – | 194097 | 419047 | – | 42699 |
| Mitigation | L-2-R | – | – | – | 2409 | 1763 | 5521 | 92588 |
| | L-2-R-always | – | – | Dummy modular multiply | 2538 | 1823 | 5773 | 121733 |
| | Power-ladder | – | – | Re-design algorithm flow | 2539 | 1883 | 5770 | 124487 |
| | Montgomery | – | – | Const. time modular multiply | 5293 | 5648 | 13636 | 95021 |
| | Parallel | – | – | Parallel multiply and square | 1809 | 2153 | 5051 | 22699 |
| | Exponent-blind | – | – | Exponent randomization blinding | 2663 | 2132 | 4381 | 100907 |
| | Base-blind | – | – | Base randomization blinding | 7226 | 12780 | 16289 | 54177 |
| Discretization | Interval-100 | – | – | Interval = 100 cycles | 1857 | 1414 | 4508 | 93115 |
| | Interval-150 | – | – | Interval = 150 cycles | 1857 | 1413 | 4507 | 93140 |
| | Interval-200 | – | – | Interval = 200 cycles | 1857 | 1415 | 4508 | 93165 |
| | Interval-300 | – | – | Interval = 300 cycles | 1857 | 1414 | 4508 | 93215 |
| | Constant | – | – | Constant time | 1866 | 1430 | 4534 | 106690 |
| Randomization | LFSR4 | – | – | Delay 0 to 15 cycles | 1860 | 1413 | 4509 | 93071 |
| | LFSR6 | – | – | Delay 0 to 63 cycles | 1868 | 1426 | 4505 | 93081 |
| | LFSR8 | – | – | Delay 0 to 255 cycles | 1872 | 1428 | 4542 | 93176 |
| | LFSR9 | – | – | Delay 0 to 511 cycles | 1875 | 1429 | 4552 | 93304 |
| | LFSR10 | – | – | Delay 0 to 1023 cycles | 1878 | 1432 | 4559 | 93560 |

### C. Discretization and Randomization Countermeasure

*1) Discretization Countermeasure:* A simple yet effect way to hide timing variation from observations is to make the total runtime for processing all messages and keys constant. This completely eliminates the timing side channel. However, it comes at a high performance penalty since all executions will have the worst case execution time. A more intelligent alternative is to quantize the RSA computation times. This enforces that the execution times are bounded to multiples of some predefined time quantum [16]. This method can help reduce computation cost but does not fully eliminate timing leakage. We refer to this quantile method [16] or bucket method [14] as the *discretization countermeasure*. It can be implemented by a simple module to delay the computation to force them to complete at certain times. This aggregates "close" completion times to a few discrete values, and thus makes it harder to discriminate between them. One can think of this as eliminating several lower bits of the timer used to measure the execution time. There is clearly a loss of information that could possibly make the attack more difficult.

In order to understand the effect of discretization countermeasure on key leakage, we use the *Sequential* design as a baseline in our implementations. The worst case execution time is used as the runtime for the *Constant* design. Then we set the execution time interval to different number of clock cycles. For example, when setting the time interval to 100 clock cycles, a runtime of 94887 clock cycles will be delayed to 94900 after discretization. In our implementations, we set the interval sizes to 100, 150, 200 and 300 clock cycles. Different discretization designs and their corresponding average runtimes are shown in the *DISCRETIZATION* group of Table I.

*2) Randomization Countermeasure:* Another frequently used technique for fuzzing the timing observation is to randomize the computation time in order to decouple the correlation between the key bits and runtime. This can be implemented by incorporating a random number generator for delay control.

In order to understand the effect of randomization countermeasure on key leakage, we also use the *Sequential* design as a baseline in our implementations. We employ linear feedback shift registers (LFSRs) to add random delay to RSA computation time. The LFSR lengths are set to 4, 6, 8, 9 and 10 bits for different implementations. Different randomization designs and their corresponding average runtimes are shown in the *RANDOMIZATION* group of Table I (*LFSRn* refers to $n$-bit LFSR).

## V. EXPERIMENTAL RESULTS

### A. Experimental Setup

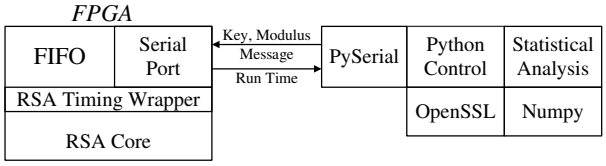


Fig. 4. Timing attack framework.

Our timing attack framework consists of three modules: test vector generation, RSA timing measurement, and statistical analysis. Figure 4 illustrates our experimental setup. We implement different RSA architectures on the *Xilinx VC707* FPGA board. We focus on 128-bit RSA cores for ease of result interpretation though the quantitative analysis method applies to RSA cores of arbitrary key length. In the test vector generation module, we use *OpenSSL* to generate RSA key pairs (key and modulus) and produce random messages with Python's pseudo-random number generator. In the statistical analysis module, we compute the variances of runtime measurements to guess each key bit using both Kocher's and sliding window timing attack methods. We calculate the mutual information

between the key bits and total runtime to quantify information leakage.

### B. Quantitative Analysis of Optimized RSA Architectures

We first analyze five different RSA cores optimized for performance (see the *OPTIMIZATION* group of Table I in Section IV-A). We collect runtime samples of 6000 different key pairs for each architecture. In order to understand the relationship between hardware architecture and key information leakage, we perform mutual information analysis using Equation (9). The results are shown in Figure 5.

From Figure 5, we can see that the *Sequential* architecture leaks about 0.7 bits of information on average while the *Unroll* architecture has 0.8 bits of leakage. Although the unroll directive allows loop parallelism in modular multiplication by using extra hardware resources and operations, it introduces more unique runtimes, increasing $H(T)$ and mutual information significantly. The *Pipeline* design reduces information leakage to 0.5 bits. *Pipeline&unroll-1* further reduces this leakage to around 0.36 bits. All these designs are implemented using one modular multiply function, so modular multiply and modular square operations execute sequentially. The pipeline and unroll directives in HLS change the structure of inner loop for these designs. The general trend is that implementation level parallelism reduces the amount of leakage.

The *Pipeline&unroll-2* design only leaks information about the highest non-zero key bit. This design has a special architecture – the synthesis tool generates two modular multipliers, one for modular multiply and the other for modular square. These two modular multipliers run in parallel, and each key bit iteration completes in the same number of clock cycles. In addition, their modular multiply and control logic times are also constant, diminishing the effect from different messages. We can see that for most of the key bits, the leakage is nearly zero but the curve increases dramatically at the end. That is due to the fact that architecture stops its execution after reaching the most significant `1` bit of the key. Thus, we can accurately determine where the highest non-zero key bit resides by observing the total runtime of the algorithm. For example, the longest runtime will have a `1` in the most significant bit. If the most significant bit is `0`, it will have a shorter runtime since the algorithm will terminate sooner. In other words, the distribution of high key bits has a significant effect on $I(k_i;T)$ by dominating the total runtime. Another way of viewing this is using Equation (2) to derive the following:

$$I(k_i;T) = H(T) - H(T|k_i) \tag{10}$$

Here, the entropy of the total runtime $H(T)$ is constant for all the key bits given the total runtime measurements; the decrease in conditional entropy $H(T|k_i)$ contributes to the increase in $I(k_i;T)$. While the decrease in the conditional entropy means that the uncertainty of total runtime $T$ given $k_i$ decreases, it also indicates that the higher key bits have a deterministic effect on the total runtime.

Note that there is sharp decrease in mutual information for higher key bits of all designs, this is due to the fact that the distribution of values for the higher key bits contributes to the decrease of $I(k_i;T)$. Although the highest key bit dominates the total runtime for each individual key, the distribution characteristics of higher key bits make $H(T|k_i)$ increase significantly. Besides, there is no information leakage for the first two key bits in any of the architectures. This is due to the fact that these two bits are the same across all of the different keys due to the requirements on how RSA keys are generated (i.e., the lowest key bit must be odd). The mutual information between a constant and another variable is constantly zero.

We conduct timing attack using Kocher's method on each design with 500 key pairs. The attack results are shown in Figure 6. Figure 6 shows that the attack success rates decrease for the higher key bits, which corresponds to the sharp decrease trend for mutual information results in Figure 5. The *Pipeline&unroll-2* design is difficult to attack; the attack success rate is around 50%, which is close to a random guess. This is not surprising given that this architecture has very little leakage as discussed earlier. Correspondingly the mutual information in Figure 5 is nearly zero. Note that the most significant bits have a higher attack success rate, which corresponds to the strong up-tick in the mutual information at the higher key bits. Visually the remainder of the attacks follow a similar trend.

Based on our analysis, the *Pipeline* design introduces 10% more area overhead while achieving 33% performance improvement and its timing channel leakage reduces to 0.5 bits. Although the *Unroll* design causes 163% more area overhead to achieve 23% performance improvement, its timing channel leakage increases to 0.8 bits. The *Pipeline&unroll-1* design has 167% more area overhead while achieving 31% performance improvement; its timing channel leakage decreases to 0.36 bits. The *Pipeline&unroll-2* design reduces the key leakage to 0 except for the highest non-zero key bit at an extremely high design cost. In the following section, we explore mitigation techniques that reduce timing channel leakage at much lower design overheads.

### C. Quantitative Analysis of Mitigated RSA Architectures

In this section, we focus on the architectures implemented with different mitigation techniques. These are discussed in Section IV-B and summarized in the *MITIGATION* group of Table I. We collect runtime samples using the same 6000 key pairs as Section V-B. Figure 7 shows the mutual information analysis results.

The *L-2-R* design has mutual information of 0.63 bits. The *Base-blind* design has mutual information of around 0.62 bits. The *Exponent-blind* design reduces key information leakage to 0.46 bits. The *L-2-R-always* and *Power-ladder* designs have mutual information of 0.11 bits and 0.18 bits, respectively. The *Montgomery* design reduces key information leakage to 0.04 bits; the synthesis tool generates an architecture with constant modular multiplication and constant control logic clock cycles, removing the effect of different messages on timing. But two modular multiply operations still run in a sequential way without changing the conditional branch structure. Different keys still result in timing variation and thus key bit leakage is not completely eliminated. The *Parallel* design reduces the key information leakage to 0.11 bits.

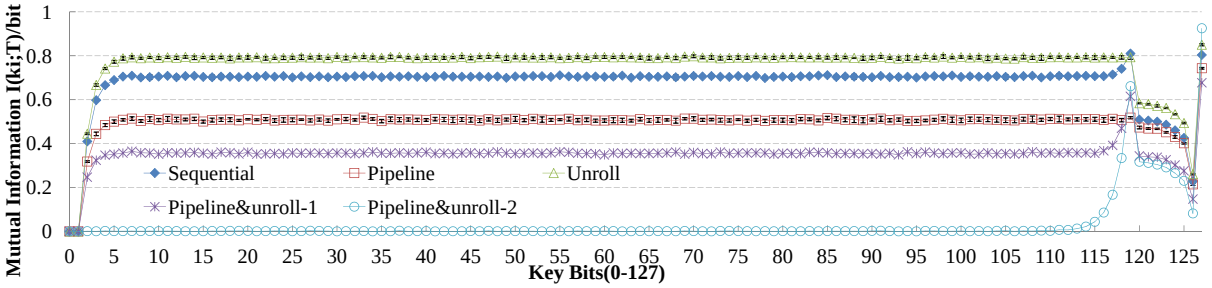


Fig. 5. Mutual information between different key bits and the total runtimes for RSA architectures generated from HLS.

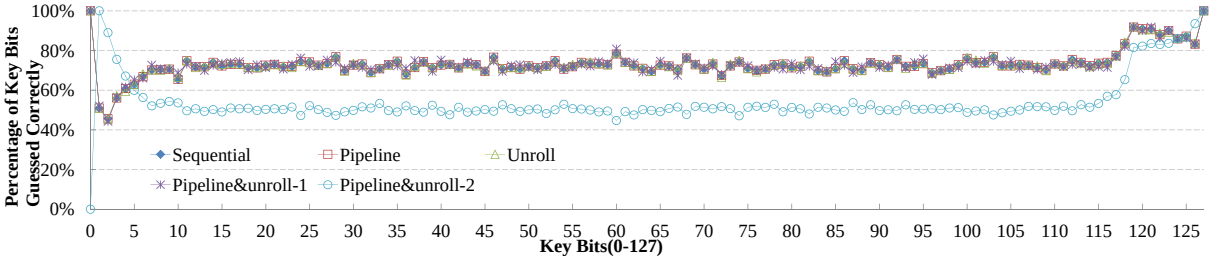


Fig. 6. Percentage of key bits guessed correctly for optimized RSA architectures, i.e., generated from HLS shown in the *OPTIMIZATION* group of Table I.

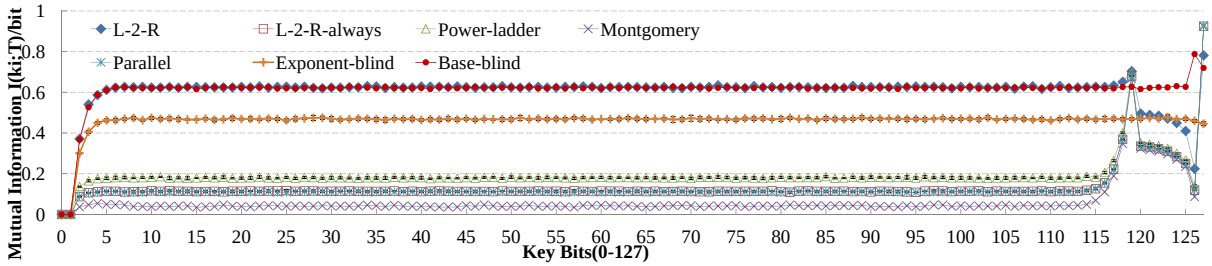


Fig. 7. Mutual information between different key bits and the total runtimes for mitigated RSA architectures.

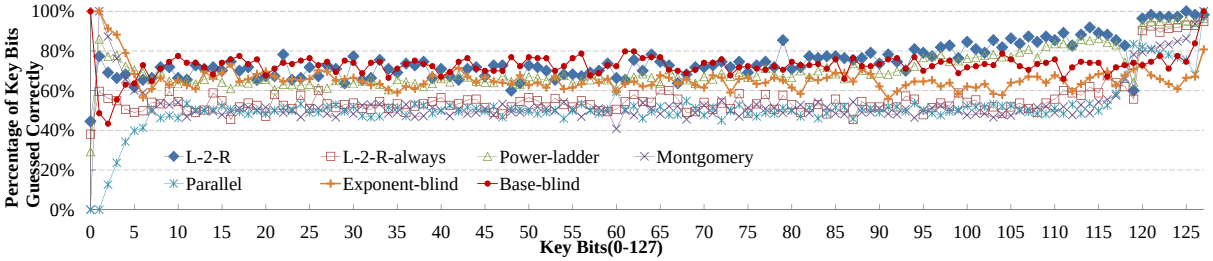


Fig. 8. Percentage of key bits guessed correctly for RSA architectures with mitigation techniques.

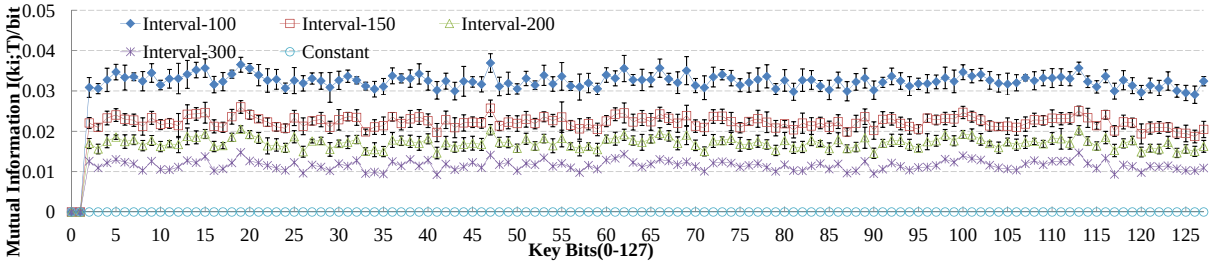


Fig. 9. Mutual information between different key bits and the total runtimes for RSA with discretization countermeasures.

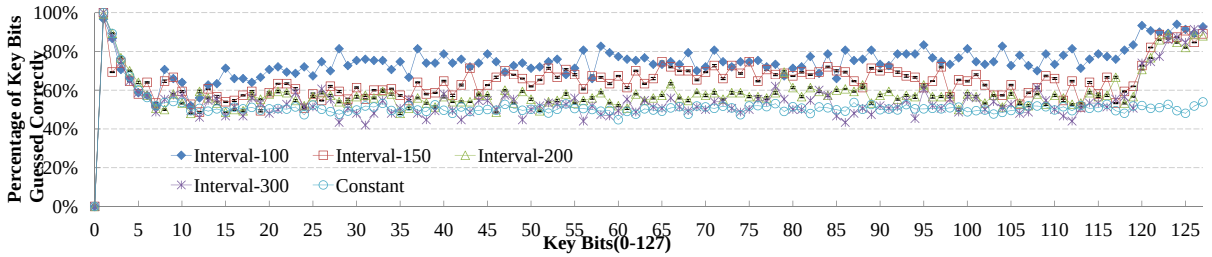


Fig. 10. Percentage of key bits guessed correctly for RSA architectures with discretization countermeasures.

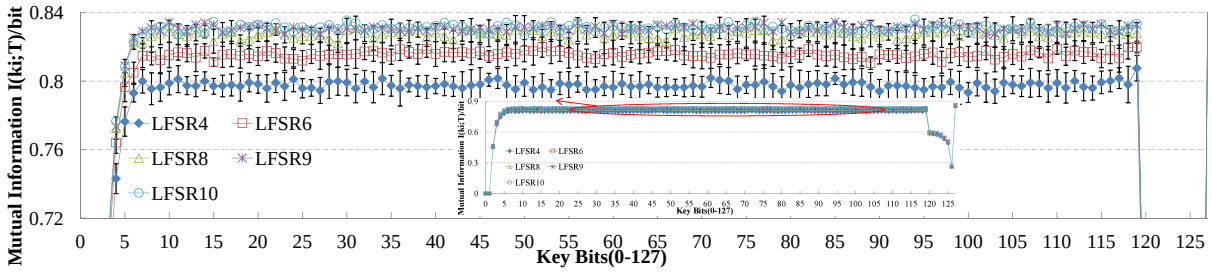


Fig. 11. Mutual information between different key bits and the total runtimes for RSA with randomization countermeasures.

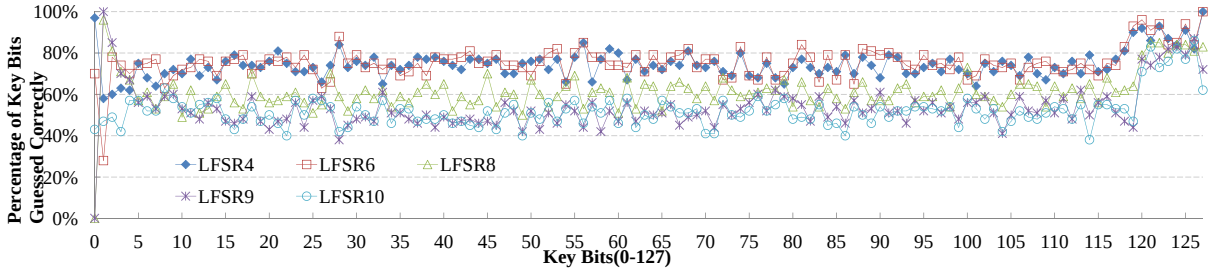


Fig. 12. Percentage of key bits guessed correctly for RSA with randomization countermeasures.

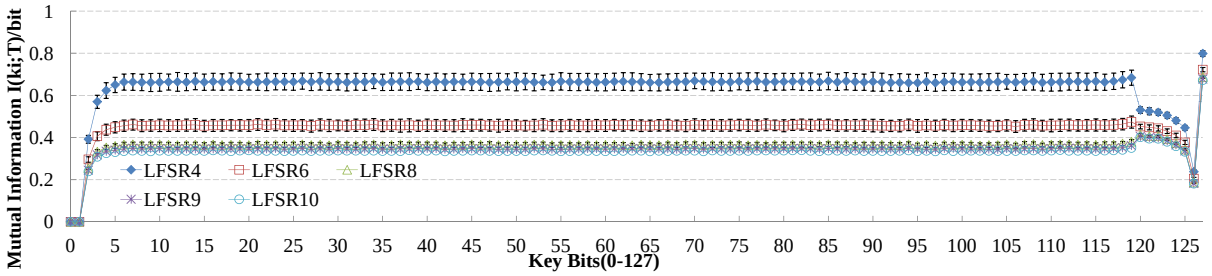


Fig. 13. Mutual information between different key bits and the total runtimes for RSA with randomization countermeasures using new attack method.

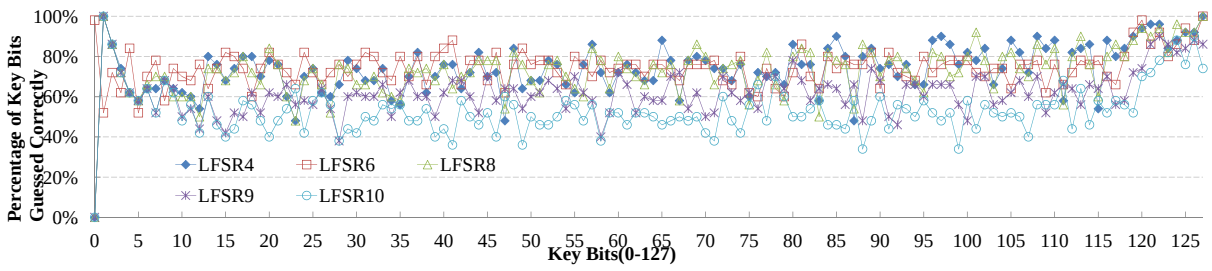


Fig. 14. Percentage of key bits guessed correctly for RSA architectures with randomization countermeasures using new attack method.

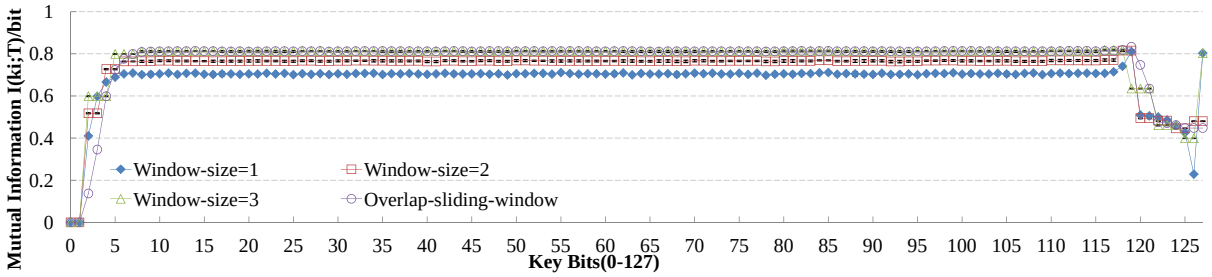


Fig. 15. Mutual information between different key bits and the total runtimes for RSA using sliding window attack methods.

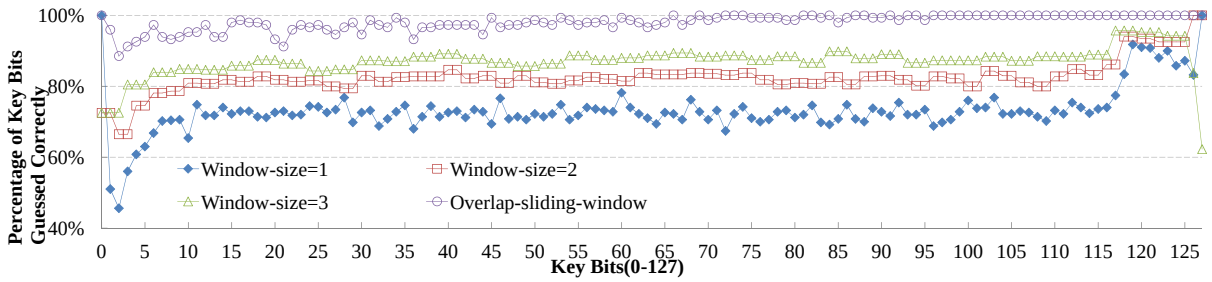


Fig. 16. Percentage of key bits guessed correctly for RSA using different sliding window attack methods.

Then we conduct Kocher's attack on these designs using the same 500 key pairs as Section V-B, the results are shown in Figure 8. The designs with mitigation techniques are more difficult to attack than the *L-2-R* design. The *Power-ladder* design is easier to attack than the *L-2-R-always* design. The *Montgomery* design and the *Parallel* design have success rates around 50%, which are less than the *L-2-R-always* and *Power-ladder* designs. The *Exponent-blind* design is harder to attack than the designs without mitigation techniques. The success rate of the *Exponent-blind* design is lower than the *Power-ladder* design, but its mutual information is higher than the *Power-ladder* design. The increase in mutual information for the *Exponent-blind* design is caused by the random number, which introduces additional uncertainty. It also indicates more powerful timing attacks probably exist for the *Exponent-blind* design (we validate that the *Non-overlapping Sliding Window* method using 3-*bit-window* can improve the success rate.). The *Base-blind* design is used to decouple the correlation between the messages and the runtimes without influencing the key leakage and the attack results significantly. Meanwhile, the *Parallel* design costs 91% area overhead as compared to the *L-2-R* design, but is 3 times faster than the *L-2-R* design while reducing mutual information to only 0.11 bits.

When comparing attack success rates with the mutual information results in Figure 7, we can see three trends. First, a higher mutual information value indicates a higher likelihood of successful attack. Second, there is an increase in the most significant bits in both mutual information results and attack success rate results for most of the mitigation designs. Third, success rate decreases in higher bits location for the *L-2-R*, *Power-ladder*, *L-2-R-always* and *Parallel* designs, which corresponds to related mutual information analysis results in Figure 7. Thus, mutual information analysis results can well reflect timing channel characteristics of mitigation designs.

### *D. Quantitative Analysis of Discretized and Randomized RSA Architectures*

*1) Discretized Architecture:* Intuitively speaking, discretization countermeasures lead to less timing variation because of coarse-grain time intervals, which reduces timing channel leakage. In this section, we focus on the architectures with discretization countermeasures summarized in the *DISCRETIZATION* group of Table I.

We use mutual information to measure information leakage with the same 6000 key pairs, and use Kocher's method to attack each design with 100 different key pairs. The experiment results are shown in Figure 9 and Figure 10. We can see that mutual information results decrease gradually while time intervals increase. And the timing attack success rates decrease, too. Because the *Constant* design leaks no information, mutual information for each key bit is 0 and the success rate is around 50%. The *Constant* design leads to 13% performance and 1% area overhead penalty to eliminate timing channel. However, discretization countermeasures improve timing channel security by sacrificing performance slightly. For example, the *Interval-300* design results in less than 1% performance penalty to achieve timing channel leakage of 0.013 bits from our mutual information analysis.

*2) Randomized Architecture:* We then use mutual information to measure information leakage with the same set of key pairs, and use Kocher's method to attack each design with 100 different key pairs. The mutual information results are shown in Figure 11 and the attack results are shown in Figure 12. However, mutual information increases while the timing attack success rate deceases with the increase of LFSR length. This is because the LFSR introduces uncertainty (i.e., additional information) to the RSA runtime. The additional uncertainty will make the design harder to attack and affect the accuracy of mutual information measurement under a limited number

of samples.

A frequently used method to attack designs protected by randomization countermeasures is to collect multiple samples under the same input condition and take the average of multiple measurements to average away the random noise. We use this method to attack the RSA architectures protected by LFSR and estimate information leakage under this new attack.

We use the same 6000 key pairs, repeat the measurement for each key 8 times and take all the runtime measurements for each key pair to calculate mutual information. Correspondingly, we apply each message 8 times and take the mean as the runtime for the message in our attack as well. The mutual information and attack results are shown in Figure 13 and Figure 14, respectively. Under this new attack method, mutual information results increase (or decrease) while the timing attack success rates increase (or decease). This change in the trend is because there is an increase in the number of samples for estimating mutual information and multiple samples for the same key pairs can now more precisely reveal the inconsistency in runtimes caused by randomization. Such inconsistency will cause collisions in mutual information estimation and lead RSA architectures with a large LFSR, which are harder to attack, to have lower mutual information. In terms of design overheads, the *LFSR9* design reduces mutual information to around 0.35 bits with only 0.3% performance and 1.6% area overhead penalty, and the *LFSR10* design reduces mutual information to around 0.34 bits with only 0.6% performance and 1.8% area penalty.

In conclusion, mitigation designs such as the *L-2-R-always* and *Power-ladder* designs balance the conditional branch structure to minimize timing difference. The *Montgomery* design implements modular multiplication in the Montgomery field to eliminate the timing difference resulting from different messages. The *Parallel* architecture reduces timing variation by increasing parallelism to mask the timing difference caused by unbalanced conditional branches. The *Discretization* countermeasures reduce the timing difference by binning the runtimes into groups. The *Exponent-blind* design and *randomization* countermeasures decouple the correlation between the key bits and the runtimes by introducing additional randomness. Our information theoretic method can also help to understand the effectiveness of different techniques on mitigating timing channels according to our analysis.

### E. Sliding Window Attack Analysis

Based on the independence assumption made for the sliding window methods in Section III-D, it is reasonable to decouple the correlation of each window on the runtimes. In this section, we present mutual information analysis and attack results for the sliding window methods. We first focus on the *Non-overlapping Sliding Window* method. Its mutual information equation is as follows:

$$I(k_i;T) = \frac{H(k_{[s:(s+l-1)]}) + H(T) - H(k_{[s:(s+l-1)]},T)}{l} \quad (i \in [s:(s+l-1)]; s = s+l; 0 \le s < w) \tag{11}$$

where $s$ denotes the key bit location, $l$ denotes the window size (it is specified to be 1, 2, 3 in this work) and $w$ denotes the key length. We can see that there is only one guess and mutual information result for each key bit in the *Non-overlapping Sliding Window* method.

We then test the *Overlapping Sliding Window* method with a window size $l$ of three. There will be three guesses and mutual information results for each key bit as shown in Section III-D. In our analysis, we need to calculate $I_1(k_s;T)$, $I_2(k_s;T)$ and $I_3(k_s;T)$ due to the overlap in windows. Then, its mutual information is calculated as Equation (12).

$$\begin{cases} I_1(k_s;T) = \dfrac{H(k_{[(s-2):s]}) + H(T) - H(k_{[(s-2):s]},T)}{3} \\ I_2(k_s;T) = \dfrac{H(k_{[(s-1):(s+1)]}) + H(T) - H(k_{[(s-1):(s+1)]},T)}{3} \\ I_3(k_s;T) = \dfrac{H(k_{[s:(s+2)]}) + H(T) - H(k_{[s:(s+2)]},T)}{3} \\ I(k_s;T) = max\{I_1(k_s;T), I_2(k_s;T), I_3(k_s;T)\} \\ \qquad (s = s+1; 0 \le s < w) \end{cases} \tag{12}$$

We employ Equations (11) and (12) to quantify the information leakage from the *Sequential* design under the sliding window attacks using the same 6000 key pairs. The results are shown in Figure 15. We can see that the mutual information result increases with the window size increasing. When using 1-*bit*, 2-*bit* and 3-*bit* windows, the mutual information results are 0.7, 0.77 and 0.81 bits, respectively. When using the *Overlapping Sliding Window* method, the mutual information is about 0.82 bits.

Then we perform timing attack based on sliding window methods using 500 different key pairs. Figure 16 shows the average accuracy for the key guesses. The results show that 1-*bit-window* method correctly guesses 73% of the key bits. The 2-*bit-window* and 3-*bit-window* methods increase attack success rates to 82% and 87%, respectively. And the *Overlapping Sliding Window* method has about 98% accuracy.

We can also see that as the mutual information results go higher with the window size increasing, the attack success rates increase too. Both mutual information and attack success rate indicate that sliding window attacks aggravate key leakage. Mutual information is capable of reflecting and assessing information leakage of the RSA timing channel in the circumstances of sliding window attacks as validated by the attack results.

### F. Correlation Analysis

To better determine the connection between mutual information and leakage, we rely on Spearman's $\rho$, as a correlation measure between mutual information (shown in Figure 5, Figure 7, Figure 9, Figure 11, Figure 13 and Figure 15) and attack success rate (shown in Figure 6, Figure 8, Figure 10, Figure 12, Figure 14 and Figure 16) for each key bit position across all the designs. In this work, we deal with each key bit position separately lest too much trouble in independence analysis between the key bits.

Spearman's $\rho$ is nonparametric, which means that the measure does not assume data comes from a particular

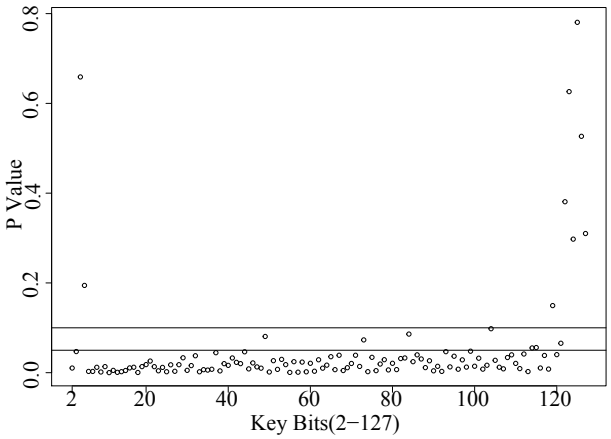


Fig. 17. Probability distribution of Spearman tests. The above line is at 0.1 significance level, the below line is at 0.05 significance level.

parametrized distribution. High correlation is achieved when one variable is a monotonic function of the other regardless of what this function may be. While significance of $\rho$ may be computed according to several distribution-based measures, we compute it using an exact permutation test for each key bit from position 2 to 127 (The least significant two key bits are removed because their values are constant). Figure 17 shows all the $p$ values of the Spearman tests. We see that 117 key bit positions (93% of the key bit positions) have $p$ less than 0.1, where at the $\alpha = 0.1$ level we reject the null hypothesis that the mutual information and success rate are uncorrelated. We see that 110 key bit positions (87% of the key bit positions) have $p$ less than 0.05, where at the $\alpha = 0.05$ level we reject the null hypothesis that the mutual information and success rate are uncorrelated. This allows us to say that for a greater value of the mutual information we have much confidence to see a greater success rate – indicating greater information leakage. In other words, both success rate and mutual information are capable of characterizing timing leakage for cryptographic hardware architectures. Based on the timing channel leakage of these architectures given different situations, mutual information is positively correlated with the attack success rate.

Given that it may not completely conform to independence condition between different key bits in reality, we can still rely on statistical tools to implement proper correlation calculation. For example, (1) using the average of the dependent data points as [17] do; or (2) analyzing each dependent key bit position separately across all the designs as this work shown; or (3) setting up statistical model such as hierarchical model or random effects model with respect to the dependence, we do not discuss this in more detail here, leaving it an open problem to discuss and solve in future.

## VI. Assess Timing Channel in Other Ciphers

Mutual information is also capable of capturing timing leakage in other ciphers such as AES and ECC. We use AES, RSA (the *Parallel* design) and ECC cores from *opencores.org* for illustration. The mutual information results are shown in Figure 18.

Timing channel in AES results from the unbalanced conditional modular reduction in the *xtime* primitive in the MixColoumns operation [18]. However, when we measure the mutual information between the key bits and runtimes using an AES core from [19], the result is 0. This indicates that there is no timing channel in this AES implementation. This is because the *xtime* primitive has been balanced to constant runtime, there is no key-dependent timing variation as in [18].

The most important computation in ECC is point scalar multiplication. The binary implementation of scalar multiplication using double-and-add algorithm has a conditional branch statement structure similar to RSA. When the key bit is one, the key bit iteration performs both point add and point double calculation; otherwise it performs point double only. The key bits have a significant effect on the runtimes of scalar multiplication. Such unbalanced conditional branch structure contributes to timing difference. For the ECC implementation from [19], we use Equation (9) to measure the mutual information between the key bits and the runtimes using 6000 keys. The results are shown in Figure 18. The mutual information is around 0.04 bits, which is pretty low. It indicates that there is a timing channel, but it can be difficult to attack. This ECC implementation has a timing channel characterization similar to the *Montgomery* RSA design. This is because the ECC implementation uses point add and point double with constant times, and the ECC input points have no effect on the timing variation [19]. So we cannot attack the ECC design by testing different input points using Kocher's method. Only the key has an effect on timing variation, which means we can still attack the ECC by testing different keys (e.g. the brute-force method).

Mutual information can assess the ECC timing channel information leakage in this test. However, a full overview of timing channel characterization in different ECC implementations requires a thorough quantitative analysis of multiple ECC designs with different parameters and mitigation techniques. We will leave it to explore in depth in future work.

## VII. Related Work

There are numerous works that use information theoretic methods to ascertain the security of a system by analyzing the behavior of the software. Denning is amongst the first to relate security and information theory [20]. McLean first describes the flow model security property [21]; it is later formalized quantitatively by Gray as an applied flow model, which relates noninterference to the maximum rate of flow between variables [22]. Clark et al. use different information theoretic measures to bound the information leaked from "while" programs [12]. Mica and Morgan use conditional entropy to calculate the channel capacity of a program [23]. McCamant and Ernst [24] present a technique to more precisely quantify how much information is revealed by the public output of C-like programs. Malacaria and Heusser [25] introduce quantitative information analysis for C code and show that the information leakage vulnerabilities in the Linux Kernel. Information theory measures, e.g., the worst case mutual information [26] and min-entropy [27], are used at the

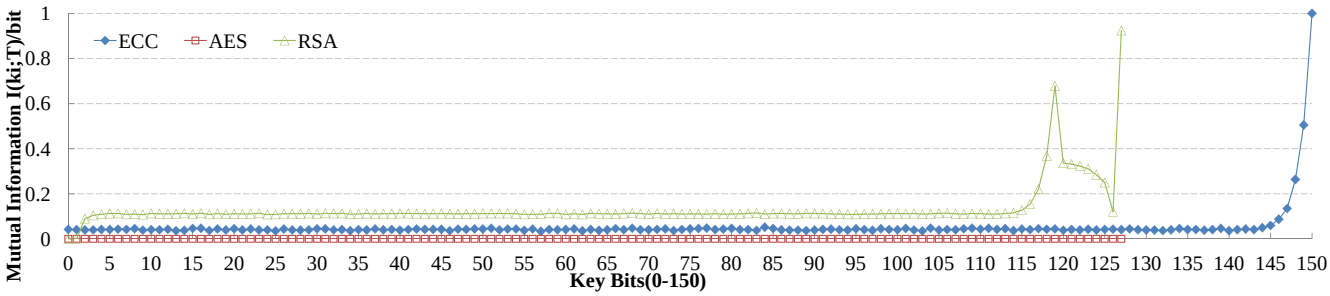


Fig. 18. Mutual information between different key bits and the total runtimes for ECC, RSA and AES implementations.

system level to determine the difficulty of breaking into the system. None of these techniques deal with hardware designs as we describe in this work.

There are numerous efforts focusing on using mutual information as a distinguisher function for side channel analysis (power, EM, and fault analysis [28], [29], [30]). Batina et al. [28] and Gierlichs et al. [31] performed a comprehensive study on mutual information analysis on the power attacks. These are inspired by Standaert et al. [32], who use mutual information to measure the amount of side-channel leakage for a cryptographic function implementation. They all use mutual information to attack the design. None of these works attempts to characterize the effects of hardware optimization, mitigation techniques and countermeasures on the timing channel as we do in this work.

Perhaps the most similar work to ours is that done by Köpf et al. [14], [33]. They provide a bound on the information leakage through a timing channel based upon the number of observations. They use conditional entropy to derive that bound. This is similar in spirit to what we do in our work in that we are trying to derive a metric for security. However, we are looking at orthogonal variables – they look at the effect of the number of measurements on the leakage, while we are trying to understand how a design itself effects the leakage and how different attacks effect the leakage.

## VIII. Conclusion

In this paper, we demonstrate the possibility of using the mutual information as a metric to quantify the amount of information a hardware architecture leaks through a timing channel. Our work reveals that the mutual information and success of the attack is correlated, i.e., a design with higher mutual information is more likely to expect higher attack success rate. Our work suggests that mutual information is a promising metric to quantify the information leakage through timing side channel.

## References

[1] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology - CRYPTO'96, Springer-Verlag Lecture Notes in Computer Science*, vol. 1109, pp. 104–113, 1996.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[3] W. Schindler, "A timing attack against RSA with the chinese remainder theorem," in *Cryptographic Hardware and Embedded Systems-CHES 2000*. Springer, January 2000, pp. 109–124.

[4] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[5] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 355–371.

[6] Y. Yarom, D. Genkin, and N. Heninger, "Cachebleed: A timing attack on OpenSSL constant time RSA," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2016, pp. 346–367.

[7] W. Schindler, "Exclusive exponent blinding is not enough to prevent any timing attack on RSA," *Journal of Cryptographic Engineering*, vol. 6, no. 2, pp. 101–119, 2016.

[8] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," in *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2015, pp. 503–516.

[9] X. Li, V. Kashyap, J. K. Oberg, M. Tiwari, V. R. Rajarathinam, R. Kastner, T. Sherwood, B. Hardekopf, and F. T. Chong, "Sapper: A language for hardware-level security policy enforcement," in *Proc. of the 19th international conference on Architectural support for programming languages and operating systems, ASPLOS 14*, March 2014, pp. 97–112.

[10] J. Oberg, S. Meiklejohn, T. Sherwood, and R. Kastner, "Leveraging gate-level properties to identify hardware timing channels," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 9, pp. 1288–1301, 2014.

[11] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[12] D. Clark, S. Hunt, and P. Malacaria, "Quantified interference for a while language," *Electronic Notes in Theoretical Computer Science*, vol. 112, pp. 149–166, 2005.

[13] T. Zhang and R. B. Lee, "New models of cache architectures characterizing information leakage from cache side channels," in *Proc. of the 30th Annual Computer Security Applications Conference*, 2014, pp. 96–105.

[14] B. Köpf and M. Dürmuth, "A provably secure and efficient countermeasure against timing attacks," in *IEEE Computer Security Foundations Symposium*. IEEE, 2009, pp. 324–335.

[15] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestr, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack," in *Smart Card Research and Applications*, ser. Lecture Notes in Computer Science, J.-J. Quisquater and B. Schneier, Eds. Springer Berlin Heidelberg, 2000, vol. 1820, pp. 167–182.

[16] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 297–307.

[17] B. Mao, W. Hu, A. Althoff, J. Matai, J. Oberg, D. Mu, T. Sherwood, and R. Kastner, "Quantifying timing-based information flow in cryptographic hardware," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE, November 2015, pp. 552–559.

[18] J.-L. Danger, D. Nicolas, G. Sylvain, and S. Youssef, "High-order timing attacks," in *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*. ACM, 2014, pp. 7–12.

[19] http://opencores.org/projects.

[20] D. E. Robling Denning, *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

[21] J. McLean, "Security models and information flow," in *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*. IEEE, 1990, pp. 180–187.

[22] J. W. Gray III, "Toward a mathematical foundation for information flow security," *Journal of Computer Security*, vol. 1, no. 3, pp. 255–294, 1992.

[23] A. McIver and C. Morgan, "A probabilistic approach to information hiding," in *Programming methodology*. Springer, 2003, pp. 441–460.
[24] S. McCamant and M. D. Ernst, "Quantitative information flow as network flow capacity," *ACM SIGPLAN Notices*, vol. 43, no. 6, pp. 193–205, 2008.
[25] J. Heusser and P. Malacaria, "Quantifying information leaks in software," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 261–269.
[26] K. Chatzikokolakis, C. Palamidessi, and P. Panangaden, "Anonymity protocols as noisy channels," in *Trustworthy Global Computing*. Springer, 2007, pp. 281–300.
[27] G. Smith, "On the foundations of quantitative information flow," in *Foundations of Software Science and Computational Structures*. Springer, 2009, pp. 288–302.
[28] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon, "Mutual information analysis: a comprehensive study," *Journal of Cryptology*, vol. 24, no. 2, pp. 269–291, 2011.
[29] O. Meynard, D. Réal, S. Guilley, F. Flament, J.-L. Danger, and F. Valette, "Characterization of the electromagnetic side channel in frequency domain," *Inscrypt*, vol. 6584, pp. 471–486, 2010.
[30] K. Sakiyama, Y. Li, M. Iwamoto, and K. Ohta, "Information-theoretic approach to optimal differential fault analysis," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 109–120, 2012.
[31] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis," in *Cryptographic Hardware and Embedded Systems–CHES 2008*. Springer, 2008, pp. 426–442.
[32] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," Cryptology ePrint Archive, Report 2006/139, 2006, http://eprint.iacr.org/.
[33] B. Köpf and D. Basin, "An information-theoretic model for adaptive side-channel attacks," in *Proc. of the 14th ACM conference on Computer and communications security*. ACM, October 2007, pp. 286–296.

**Baolei Mao** is currently pursuing the Ph.D. degree from the School of Automation, Northwestern Polytechnical University. From 2013 to 2015, he was a visiting graduate student in Kastner Group in University of California, San Diego. He is a recipient of Internet Security Graduate Student Award from China Internet Development Foundation.

His current research interests include timing and power side channel analysis, hardware information flow analysis and formal verification.

**Wei Hu** received the Ph.D. degree in control theory and control engineering from Northwestern Polytechnical University, Xian, Shaanxi, China, in 2012.

He is currently an Associate Professor with the School of Automation, Northwestern Polytechnical University, China. His research interests include hardware security, logic synthesis, and formal verification, reconfigurable computing and embedded systems.

**Alric Althoff** received the B.S. degree in cognitive science and mathematics from the University of California, San Diego (UCSD), La Jolla, CA, USA, in 2013. He is currently working toward the Ph.D. degree in computer science at UCSD.

His current research interests include statistics, signal processing, and machine learning, focusing on hardware security, experimental design in environments where individual experiments are costly, and high-throughput data analysis.

**Janarbek Matai** received the B.S. degree in computer science from the Mongolian University of Science and Technology, Mongolia, in 2004, and the Master of Computer Science degree from the Korea Advanced Institute of Science and Technology, South Korea, in 2007. He received Ph.D. degree from University of California, San Diego, in 2015.

His current research interests include high-level synthesis, graphics processing units, reconfigurable computing, and application specific processors.

**Yu Tai** is currently pursuing the Ph.D. degree with the School of Automation, Northwestern Polytechnical University, Xian, Shaanxi, China. From 2015 to 2016, he is a Visiting Graduate Student with the Department of Computer Science and Engineering, University of California, San Diego.

His current research interests include hardware security, logic synthesis and optimization in hardware information flow.

**Dejun Mu** received the Ph.D. degree in control theory and control engineering from Northwestern Polytechnical University, Xian, Shaanxi, China, in 1994.

He is currently a Professor with the School of Automation, Northwestern Polytechnical University, China. His current research interests include control theories and information security, including network information security, application specific chips for information security, and network control systems.

**Timothy Sherwood** is a Professor of Computer Science and Associate Vice Chancellor for Research at UC Santa Barbara where he specializes in the development of processors exploiting novel technologies (e.g. plasmonics and memristors), provable properties (e.g. information flow security and correctness), and hardware-accelerated algorithms (e.g. high-throughput scanning and new logic representations). In 2013 he co-founded Tortuga Logic to bring rich security analysis to the hardware and embedded system design processes. He is the recipient of the Northrop Grumman Teaching Excellence Award, the SIGARCH Maurice Wilkes Award, co-author of 11 award winning articles, and an ACM Distinguished Scientist.

**Ryan Kastner** received the Ph.D. degree in computer science from the University of California, Los Angeles, CA, USA, in 2002.

He is currently a Professor with the Department of Computer Science and Engineering with the University of California, San Diego, CA, USA. His current research interests include hardware acceleration, hardware security, and remote sensing.

Prof. Kastner is the Co-Director of the Wireless Embedded Systems Master of Advanced Studies Program. He also co-directs the Engineers for Exploration Program.