

Novel PUF-based Error Detection Methods in Finite State Machines ^{*}

Ghaith Hammouri, Kahraman Akdemir, and Berk Sunar

Worcester Polytechnic Institute
100 Institute Road, Worcester, MA 01609-2280
{`hammouri, kahraman, sunar`}@wpi.edu

Abstract. We propose a number of techniques for securing finite state machines (FSMs) against fault injection attacks. The proposed security mechanisms are based on physically unclonable functions (PUFs), and they address different fault injection threats on various parts of the FSM. The first mechanism targets the protection of state-transitions in a specific class of FSMs. The second mechanism addresses the integrity of secret information. This is of particular interest in cryptographic FSMs which require a secret key. Finally, the last mechanism we propose introduces a new fault-resilient error detection network (EDN). Previous designs for EDNs always assume resilience to fault injection attacks without providing a particular construction. The PUF-based EDN design is suitable for a variety of applications, and is essential for most fault resilient state machines. Due to the usage of PUFs in the proposed architectures, the state machine will enjoy security at the logical level as well as the physical level.

Key words: Fault-resilience, state-machines, adversarial-faults, PUF

1 Introduction

Over the last decade, side-channel attacks drew significant attention in the field of cryptography. This class of attacks mainly depend on leaking secret information through implementation specific side-channels. Various attack mechanisms and countermeasures have been published in this domain, yet this is still an active field of research both in academia and industry. Roughly speaking, side-channel attacks can be classified into two main categories which are passive and active attacks.

Passive attacks depend on observing and analyzing the implementation specific characteristics of the chip. Power and timing analysis are the most powerful and common attacks in this branch. In these attack mechanisms, power and timing information of the cryptographic hardware is measured/observed at various steps and a following statistical analysis reveals information about the secret in

^{*} This material is based upon work supported by the National Science Foundation under NSF Grants No. CNS-0831416 and CNS-0716306.

the system [24, 25]. On the other hand, in an active attack the adversary actually changes the state/value of specific data in the chip in order to gain access to the secret information. In other words, the attacker injects faults to specific parts of an integrated circuit (IC) through side-channels and uses the injected flaw to gain access to the secret information such as the cryptographic key. Boneh et al. [8] demonstrated the effectiveness of these attacks on breaking otherwise impenetrable cryptographic systems. Some of the most crucial active fault attack mechanisms discussed in the literature are external voltage variations, external clock transients, temperature variations and bit-flips using highly dense optical laser beams [38, 36]. For more information on the details of side-channel attacks the reader is referred to [4, 31].

Due to their strength, side-channel attacks create a serious and crucial threat to the existing cryptographic infrastructure. Especially active attacks, even though harder to introduce, are very effective. Various counter-measures have been suggested to counter act this class of attacks and to provide secure/fault-resilient cryptographic hardware designs. Most of the solutions proposed to secure against fault injection attacks utilize some form of a concurrent error detection (CED) mechanism [6, 22, 12]. Non-linear error detection codes, a version of CED, arose as the most effective of these defense mechanisms. These codes provide a high level of robustness (the maximum error masking probability is minimized over all possible error vectors) with a relatively high hardware cost [20, 28, 27, 19, 21].

Another countermeasure proposed against side-channel attacks is the dual-rail encoding. In these schemes, a data value is represented by two wires. In this case, two out of four states represent the data and the two extra states which are “quiet” and “alarm” can be used for security purposes. Consequently, utilizing the dual railing for the cryptographic designs can potentially provide security against active and passive side-channel attacks [39, 9, 42].

A different interesting approach to protect against active attacks in authentication schemes are Physically Unclonable Functions (PUFs) [10, 33, 37]. A PUF is a physical pseudo-random function which exploits the small variances in the wire and gate delays inside an integrated circuit (IC). Even when the ICs are designed to be logically identical, the delay variances will be unique to each IC. These variances depend on highly unpredictable factors, which are mainly caused by the inter-chip manufacturing variations. Hence, given the same input, the PUF circuit will return a different output on different chipsets [30]. Additionally, if the PUFs are manufactured with high precision, any major external physical influence will change the PUF function and therefore change the output. These features are indeed very attractive for any low cost scheme hoping to achieve tamper-resilience against active attacks.

In this paper, we focus our attention on active attacks and present a CED design based on PUFs. As we will see in the next section we focus our attention on error detection in the control logic of hardware implementations. The remainder of this paper is organized as follows: Section 2 discusses our motivation and contributions. Section 3 introduces the necessary background on PUF circuits. Our PUF-based approach to secure known-path state machines is discussed in

Section 4. Next, the key integrity scheme utilizing PUFs is described in Section 5. In Section 6, a PUF-based secure error detection network (EDN) is presented and conclusions are summarized in Section 7.

2 Motivation

Almost all of the research related to CED focused on protecting the datapath of the cryptographic algorithms. However, control unit security against active fault attacks was mostly neglected and therefore has led to a significant security breach in many cryptographic hardware implementations. A literature review indicates that there is not much work done about this topic, except some simple single-bit error detection and correction scheme descriptions in the aerospace and communications areas [26, 5]. These schemes are proposed against naturally occurring faults (for example due to radioactive radiation and mostly single event upsets) on the control units and are not nearly sufficient when a highly capable adversary is considered. For example, the encryption states in a finite state machine (FSM) can be bypassed and the secret information can be easily revealed by this type of an adversary. Similarly, the state which validates the login information can be skipped and the attacker can directly impersonate a valid user with a limited effort. At the end of the day, the states in a control unit are implemented using flip-flops, which are vulnerable to bit flips and fault injection attacks.

After observing this gap in control unit fault-tolerance, Gaubatz et al. [13] conducted an initial study on this issue. In their paper, the authors describe an attack scenario on the control structure of the Montgomery Ladder Algorithm. In this scenario the adversary can easily access to the secret information by actively attacking to the controller segment of the hardware. As a solution to this problem, the authors propose applying linear error detection codes on the control units of cryptographic applications. Although the presented approach is quite interesting, it can only provide a limited level of security and robustness.

Our Contribution: In this paper, we present a high level description of a technique which uses PUFs to secure a class of finite state machines against fault injection attacks. Our proposal offers a two-layer security approach. In the first layer the PUF's functionality is used to produce a checksum mechanism on the logical level and in the second layer, the intrinsic sensitivity of the PUF construction is used as a protection mechanism on the physical level. An injected error has a high probability of either causing a change in the checksum, or causing a change in the PUF characteristics, therefore signaling an attack. With this dual approach our proposal opens an interesting area of research which explores hardware specific features of state machines. The proposed design integrates with a finite state machine in three different ways: 1) It provides a checksum mechanism for the state transitions 2) It provides key integrity protection for any secrets used by the state machine 3) It provides a novel fault-resilient implementation of the error detection network. Our work here is the first study on utilizing PUFs to secure the control logic in hardware implementations. In addition, the utiliza-

tion of PUFs proved to be extremely suitable when the hardware resources are limited.

3 Physically Unclonable Functions

A PUF is a challenge-response circuit which takes advantage of the interchip variations. The idea behind a PUF is to have the same logical circuit produce a different output depending on the actual implementation parameters of the circuit. The variations from one circuit implementation to another are not controllable and are directly related to the physical aspects of the fabrication environment. These aspects include temperature, pressure levels, electromagnetic waves and quantum fluctuations. Therefore, two identical logical circuits sitting right next to each other on a die in a fabrication house might still have quite different input-output behavior due to the nature of a PUF.

The reason one might seek to explore such a property is to prevent any ability to clone the system. Additionally, because of the high sensitivity of these interchip variations it becomes virtually impossible for an attacker to accurately reproduce the hardware. Another major advantage of the PUF's sensitivity is to prevent physical attacks on the system. Trying to tap into the circuit will cause a capacitance change therefore changing the output of the PUF circuit. Removing the outer layer of the chip will have a permanent effect on these circuit variables and again, it will change the output of the PUF circuit. Briefly, we can say that a well-built PUF device will be physically tamper-resilient up to its sensitivity level. Multiple designs have been proposed in the literature to realize PUFs. Optical PUFs were first introduced in [34]. Delay-based PUFs or more known as silicon PUFs were introduced in [10]. Coating PUFs were introduced in [40]. More recently, FPGA SRAM PUFs were introduced in [14]. Surface PUFs were proposed in [33, 41] and further developed in [37]. In general, so far the usage of PUFs has focused on the fact that they are unclonable. In this paper we focus our attention to the delay-based PUF first introduced in [10].

A delay based PUF [10] is a $\{0, 1\}^n \rightarrow \{0, 1\}$ mapping, that takes an n -bit challenge (C) and produces a single bit output (R). The delay based PUF circuit consists of n stages of switching circuits as shown in Figure 1. Each switch has two input and two output bits in addition to a control bit. If the control bit of the switch is logical 0, the two inputs are directly passed to the outputs through a straight path. If on the other hand, the control bit to the switch is 1, the two input bits are switched before being passed as output bits. So based on the control bit of every switch, the two inputted signals will take one of two possible paths. In the switch-based delay PUF, there are n switches where the output of each switch is connected to the input of the following switch. At the end, the two output bits of the last switch are connected to a flip-flop, which is called the *arbiter*. The two inputs to the first of these switches are connected to each other, and then connection is sourced by a pulse generator.

The delay PUF can be described using the following linear model [10, 15],

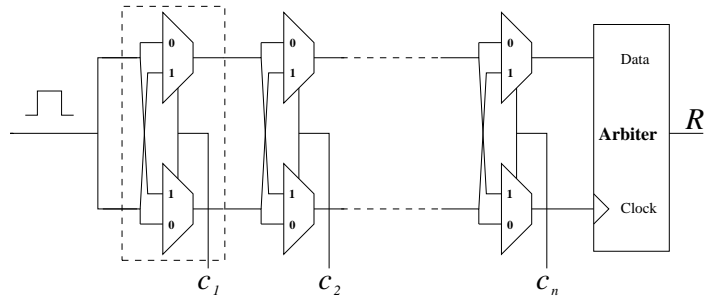


Fig. 1. A basic delay based PUF circuit

$$R = \text{PUF}_Y(C) = \text{sign} \left(\sum_{i=1}^n (-1)^{p_i} y_i + y_{n+1} \right) . \quad (1)$$

where $Y = [y_1 \dots y_{n+1}]$ with y_i as the delay variation between the two signal paths in the i^{th} stage and y_{n+1} captures the setup time and the mismatch of the arbiter. $\text{Sign}(x) = 1$ if $x > 0$, and 0 if $x \leq 0$. $p_i = c_i \oplus c_{i+1} \oplus \dots \oplus c_n$, where c_i is the i^{th} bit of C . Note that the relation between $P = [p_1 \dots p_n]$ and $C = [c_1 \dots c_n]$ can be described using the equation $(P = UC)$. The strings C and P are represented as column vectors, U is the upper triangular matrix with all non-zero entries equal to 1 and the matrix multiplication is performed modulo 2. Equation 1 captures the ideal PUF response. However, due to race conditions which will sometimes cause the two signals inside the PUF paths to have very close delays, the output of the PUF will sometimes be random. We refer to these random outputs as *metastable* outputs. This happens with a certain probability depending on the sensitivity of the arbiter. For typical flip-flops a 64-bit PUF will have about 1 metastable output for every 1000 outputs [29].

The variables y_i capture the secret maintained in the hardware, and which cannot be measured directly. These variables are usually assumed to be independent with each following a normal distribution of mean 0 and some variance which can be assumed to be 1 without loss of generality [30]. We note here that the independence of these variables will highly depend on the manufacturing process. For example in an FPGA implementation it is much harder to get almost independent y_i variables. In an ASIC implementation the y_i variables seem to be closer to independence. However, to simulate an independent response one might average over multiple independent challenges. In this study, we will work under the assumption that the y_i variables are independent.

With the independence assumption one can derive the probability distribution of two inputs $C^{(1)}$ and $C^{(2)}$ producing the same PUF output over all possible outputs as follows,

$$\text{Prob}[\text{PUF}_Y(C^{(1)}) = \text{PUF}_Y(C^{(2)})] = 1 - \frac{2}{\pi} \arctan \left(\sqrt{\frac{d}{n+1-d}} \right). \quad (2)$$

where d is the Hamming distance between $P^{(1)} = UC^{(1)}$ and $P^{(2)} = UC^{(2)}$. For the full derivation the reader is referred to [16, 15]. This relation carries the important fact that the correlations in the PUF output are solely dependent on the Hamming distance.

It is important to mention that due to the linear nature of the PUF circuit, given a number of challenge-response pairs (C, R) , an attacker can use linear programming [35, 32, 10] to approximate for the unknown vector Y . To solve this problem we have one of two options. First, hide the output R such that it is not accessible to an adversary. Our second option is to use non-linearization techniques such as the feed-forward scheme presented in [10, 29]. For simplicity we will assume that the output R is not available to an adversary. This only means that the attacker cannot read R , but he still can inject a fault. We will shortly see from the coming sections that this is quite a reasonable assumption.

Adversarial Fault Model: In this paper, we do not put a limit on the possible fault injection methods that can be used by the attacker. In other words, the adversary can utilize any attack mechanism on the device in order to successfully inject a fault. The injected fault on a specific part of the circuit manifests itself at the output as an erroneous result. Consequently, the error e becomes the difference between the expected output x and the observed output $\tilde{x}=x+e$. This error can be characterized either as logical or arithmetic depending on the functions implemented by the target device. If the target area of the device is mostly dominated by flip-flops, RAM, registers, etc. then using the logical model, where the error is expressed as an XOR operation ($\tilde{x}=x \oplus e$), is more appropriate. If on the other hand, the targeted region of the device is an arithmetic circuit, then it is more useful to use the arithmetic model where the error may be expressed as an addition with carry ($\tilde{x}=x+e \bmod 2^k$, where k is the data width). In this paper, we are mostly concerned with secure FSMs and key storage. As a result, it is more appropriate to use the logical fault model. It is also important to note that since the analysis conducted in this paper does not assume attacker’s inability to read the existing data on the circuit before injecting a fault, overwriting and jamming errors can also be modeled as logical error additions. In addition, note that one additional assumption for the fault model of Section 5 is described by Assumption 1.

4 Securing Known-Path State Machines

In this section we address the security of state machines against adversarial fault attacks. We focus our attention to the state machines which are not dependent upon input variables. Such machines are quite common in cryptographic applications which typically contain a limited number of states and perform functions

Algorithm 1 Always multiply Right-to-Left Binary Exponentiation Algorithm [17]

Require: $x, e=(e_{t-1}, \dots, e_0)_2$
Ensure: $y = x^e$

$R_0 \leftarrow 1$	INIT
$R_1 \leftarrow x$	LOAD
for $i = 0$ upto $t - 1$ do	
$b = 1 - e_i$	
$R_b \leftarrow R_b^2$	SQUARE
$R_b \leftarrow R_b \cdot R_{e_i}$	MULTIPLY
end for	
$y \leftarrow R_0$	RESULT

which require a long sequence of states. We now formally define the class of state machines which we address in this section.

Definition 1. *A known-path state machine, is a state machine in which state transitions are not dependent upon the external input. The state-sequence which the state machine goes through is known beforehand, and can be considered a property of the state machine.*

An example of an algorithm that can be implemented with a known-path state machine is the “always multiply right-to-left binary exponentiation” [17] which is shown above. The associated state diagram for this algorithm is shown in Figure 2 with a possible fault injection attack (indicated by the dotted line). As can be observed, once the start signal is received, the transitions will follow a specific pattern and are independent from any kind of input except i which is a predetermined value. Another example of a known-path state machine is the “Montgomery ladder exponentiation” which is commonly used for RSA signature generation [18]. The PUF based security mechanism which we describe in this section defines a generic approach to secure this class of state machines. We now rigorously define our approach and derive the probability of detecting an injected error.

Let F be a known-path state machine with m states. We refer to the known-path of states which the state machine goes through in a full operation as the *state-sequence* and we denote it by S_Ω . Here, Ω represents the encoded states in the state-sequence observed by F . We define f to be the encoding function for our states. The function f is also assumed to produce a binary output. Let n be the bit size of the output of f and k the number of states in Ω . So for example, if F enters the state-sequence $s_1, s_2, s_3, s_4, s_3, s_4, s_5$ then $\Omega = [f(1), f(2), f(3), f(4), f(3), f(4), f(5)]$. If the encoding scheme is a simple binary encoding then $\Omega = [001, 010, 011, 100, 011, 100, 101]$, $n = 3$ and $k = 7$.

With the above definitions our proposal’s main goal becomes to finger-print the state-sequence S_Ω . The way we achieve this is by adding a PUF circuit to the state machine logic. The straightforward idea of the scheme works as follows: at initialization time the circuit calculates the PUF output for each of

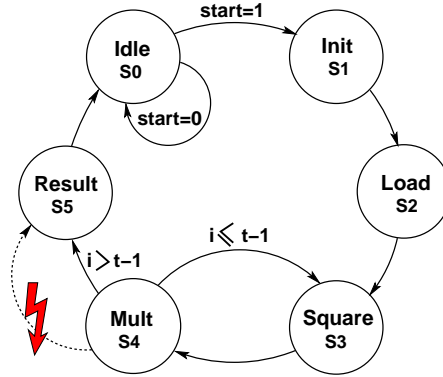


Fig. 2. State Diagram Representation of Left-to-Right Exponentiation Algorithm with Point of Attack

the encodings in Ω . This output which we label ω is then securely stored for future comparisons. The i^{th} bit of ω which we label ω_i is calculated as

$$\omega_i = \text{PUF}_Y(\Omega(i))$$

where $\Omega(i)$ is the i^{th} entry in Ω . This equation means that an n -bit PUF needs to be utilized, and that ω will be a k -bit string. This straightforward approach captures the essential idea of the proposal. However, there are problems with the efficiency of this approach. One can imagine a simple state machine going into a 3-state loop for 1000 cycles. This would mean that ω contains at least 3000 bits of a repeating 3-bit sequence. If secure storage is not an issue, or if k is a small number, then the straightforward approach should suffice. However, when a state machine is expected to have large iterations over various cycles a different approach should be explored.

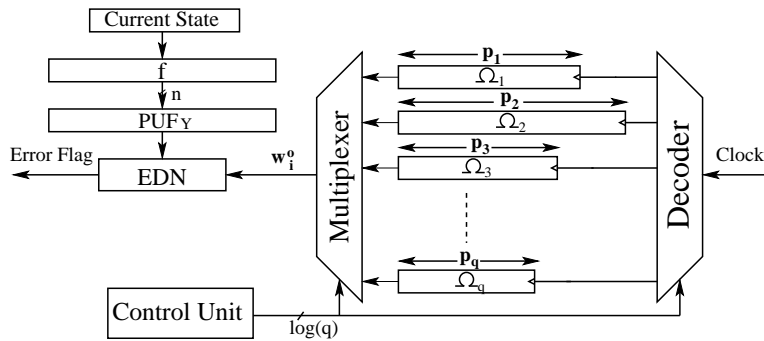


Fig. 3. PUF-based circuit for protecting FSMs

To solve this problem we take a deeper look into the state-sequences that we expect to observe in the known-path state machines. Such state-sequences can be broken into q different sub-sequences each of which contain p_i states and is repeated for c_i cycles. We can now rewrite the overall state-sequence as

$$\Omega = [\{\Omega_1\}^{c_1} | \{\Omega_2\}^{c_2} | \dots | \{\Omega_q\}^{c_q}] ,$$

where $|$ stands for concatenation of sequences, and $\{\Omega_i\}^{c_i}$ indicates the repetition of the sub-sequence Ω_i for c_i times. Note that $k = c_1p_1 + c_2p_2 + \dots + c_qp_q$. With the new labeling, we can see that the checksum does not need to be of length k . It should suffice for the checking circuit to store the constants c_i and p_i in addition to the bits $\omega_{i,j} = \text{PUF}_Y(\Omega_i(j))$ for $i = 1 \dots q$ and $j = 1 \dots c_i$. We can write

$$\omega = [\omega_1 | \omega_2 | \dots | \omega_q] ,$$

where ω_i will contain p_i bits. With only having to store the ω_i strings, the scheme will be efficient in terms of storage. We next explain how the checking circuit works.

As can be seen in Figure 3, the checking circuit stores each of the ω_i strings in a separate shift register with the output (the most significant bit) connected to the input (the least significant bit) of the register. The outputs of the shift registers are also connected to a multiplexer, whose $\log(q)$ select signals connected to a small control unit. The control unit's main task is to maintain a counter C which will indicate how far along the state-sequence is the state machine, therefore generating the select signals for the multiplexer. When $C \leq c_1p_1$ the first register's output is selected. Once the counter exceeds c_1p_1 the control unit will select the second register, and will maintain the same output until the counter reaches $c_1p_1 + c_2p_2$. The control unit will essentially chose the i^{th} register as long as $c_1p_1 + c_2p_2 + \dots + c_{i-1}p_{i-1} < C \leq c_1p_1 + c_2p_2 + \dots + c_ip_i$. The checking circuit continues in this fashion until the counter reaches $k = c_1p_1 + c_2p_2 + \dots + c_qp_q$ at which point the counter resets to zero since the state machine will be back to its initial state. We will label the output of the multiplexer at i^{th} state of the state-sequence as ω_i^o . For the registers to generate the right output, the select signals produced by the control unit also need to be fed into a decoder which will produce the clock signals of the registers. The input of the decoder will be the master clock signal, and the outputs of the decoder will be connected to the clock inputs of the shift registers. Note that these signals will only be high when the corresponding register is being used, therefore causing the register to shift accordingly.

At every clock cycle the current check bit $x_i = \text{PUF}_Y(f(S_i))$ is generated, where S_i is the current state of the state machine. The checking circuit will verify the condition $\omega_i^o = x_i$. Whenever this condition is violated the checking circuit can issue a signal to indicate a fault injection. We have mentioned earlier that the output of a PUF circuit will not be consistent for a certain percentage of the inputs. This percentage will set a tolerance threshold labeled L for the checking circuit. If the number of violations detected by the checking circuit is

more than L , the checking circuit can signal an attack, therefore halting the circuits operation.

To calculate the probability of an attack actually being detected, we note that the PUF output is uniform. A fault injected by the attacker will change the current state, and will consequently change the following states. We label the states in the fault-free sequence S_Ω as ideal states, and we label the states which are different as a result of the fault injection as faulty states. The fault-free sequence and the new faulty sequence will have t different states, $t \leq k$. We are interested in calculating the probability of the new faulty states actually yielding a different PUF output than that of the ideal states. Equation 2 shows that when the Hamming distance between the two PUF inputs is about $n/2$, this probability is 0.5. With this in mind, we can choose the encoding function f and the size of its output n such that the encodings of any two states have a Hamming distance of $n/2$. Even more efficiently, if the encoding is assumed to be secret, the Hamming distance between the encoded state vectors would be averaged over all possible encodings, therefore also yielding an effective Hamming distance of $n/2$ between any two state vectors. In either case, the probability of a faulty state generating the same PUF output as an ideal state will effectively be 0.5. With these factors, we can expect the detection probability of an injected fault which causes a total of t state changes to be

$$P_t = 1 - 2^{-(t-L)} \ .$$

Naturally, t is assumed to be larger than L since otherwise the detection probability would be zero. If the fault injected causes a small number of state changes t , this probability will not be sufficient to secure the system. Although it is expected that an injected fault will cause a large number of state changes, for completeness we next handle the case when t is small.

We propose two approaches to solve this problem. The first is to utilize a number of PUF circuits each of which storing a separate array of checksums. And the second is to use a single PUF but calculate the check bits for different encoding functions of the states. Essentially one can use a single encoding and then apply a permutation to generate a variant encoding. Whether we use d PUFs or we use d different encodings, in either case we will be adding d -levels of check bits. Regardless of which of the two approaches we use the detection probability of a fault causing t state changes will be

$$P_t = 1 - 2^{-(td-L)} \ .$$

Using error control techniques for the PUF circuit such as majority voting [29], the error rate in the PUF output can be reduced to as low as 1 in 1000. This means that for state machine where $k < 1000$ states, the probability of error detection becomes $P_t = 1 - 2^{1-td}$. Naturally, this probability does not take into account the probability of inducing a change in the internal PUF parameters. Such a change will have an effect on the PUF output and will therefore increase the error detection probability.

In order to estimate the hardware overhead incurred by the proposed error detection mechanism, we carry out the following analysis. The number of flip-flops required for storing the checksums will be equivalent to the number of flip-flops used in the current state register. As mentioned earlier, the counter C constitute the main part of the control unit, which is also the case for the state machine. Therefore, we can argue that the size of the control unit and the checksum storage will be approximately the same as the state machine, implying a 100% overhead.

The encoding function f will typically have an output size which is on the order of the total number of states m . Consequently, we can assert that the function f will on average use about $2m$ combinational gates. The same applies to the PUF circuit which will also require about $2m$ gates. Finally, the size of the decoder and the multiplexer shown in Figure 3 is expected to be on the order of $\log(q)$ gates. Although q can be of any size, in a typical state machine q will not be larger than 2^m . Hence, the number of gates associated with the multiplexer and the decoder will be about $2m$. Adding these numbers results in a total gate count of $6m$. This is the same number of gates used by the current state register (Note that m flip-flops are approximately composed of $6m$ universal gates). In general, it is safe to assume that the current state register will consume approximately 50% of the total state machine area, which implies an area overhead of 50%.

As a result, the total area overhead introduced by the proposed error detection scheme will be approximately 150%, with a high error detection rate even against strong adversaries. When compared to the simpler error detection schemes such as Triple Modular Redundancy (TMR) and Quadruple Modular Redundancy (QMR) (which only replicate the existing hardware, implement the same function concurrently, and do a majority voting to check if an error has been injected), the proposed scheme provides a higher level of security even against advanced adversaries because an attacker can simply inject the same error to all replicas of the original hardware and mask the error in these detection schemes. The area overhead associated with these trivial detection mechanisms will be at least 200% for TMR and 300% for QMR which is also higher than the overhead of the proposed mechanism. As a comparison to a more advanced error detection scheme, the study conducted by Gaubatz et al. [13], which utilizes linear codes for error detection, reports an area overhead of more than 200%. However, their fault model assumes weak adversaries and the error detection scheme becomes vulnerable against strong attackers. It is important to note that, finite state machines usually constitute a very small part of the entire circuit. Therefore, although the reported area overhead might appear to be large, the effective increase in the overall area is reasonable. To sum up, PUF-based error detection mechanism discussed in this paper accomplishes a higher level of security with a reduced area overhead.

5 Key Integrity

In [7], Biham and Shamir extended the fault injection attacks to block-ciphers and reported that they can recover the full DES key of a tamper-proof encryptor by examining 50 to 200 cipher-texts. In their paper, they also described a method to break an unknown (unspecified) cryptosystem by utilizing the asymmetric behavior associated with the used memory device. Basically, their fault model assumes that the applied physical stress on the memory device, which contains the key bits, could only cause one to zero transitions¹. Using this attack, the secret key can be obtained using at most $O(n^2)$ encryptions. Similarly, Kocar [23] reports a method to estimate the key bits of a cryptographic device by employing the charge shifting characteristic of EEPROM cell transistors after baking. In addition, in [3] authors describe an EEPROM modification attack where they can recover the DES key by overwriting the key bits one by one and checking for parity errors. Same authors, in [2], also discuss example attacks on PIC16C84 microcontroller and DS5000 security processor in which security bits can be reset by modifying the source voltages. The key overwrite attacks also constitute a crucial risk on smart cards where the key is stored inside the EEPROM. To summarize, fault injection attacks on secret keys stored on-chip memory pose a serious threat in many cryptographic hardware implementations.

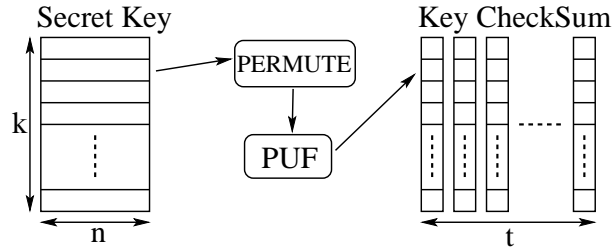


Fig. 4. Key integrity check using PUF

In this section, we propose utilizing PUFs as a solution to this important problem. The main idea here is similar to that of the previous section. Basically, we generate a secret key fingerprint or checksum for the correct secret key using a PUF circuit. As outlined earlier, the checksums are assumed to be stored secretly while allowing fault injection. Regularly, the checking circuit can check and verify the integrity of the key. If the checksum value for the current key does not match the checksum value for the correct secret key, this can be interpreted as an error injection to the key. As a result, an error message can be issued and the secret data can be flushed or the device can be reset to prevent any kind of secret leakage. This mechanism is briefly shown in Figure 4. This figure shows

¹ This one-way characteristic can also cause zero to one transitions depending on the asymmetry of technology used to fabricate the memory device.

part of the memory which contains a secret key of size $k \times n$. Each row of this key block is labeled r_i and is treated as an input to the PUF circuit. If the rows are directly fed to the PUF circuit, an attacker can carefully choose his errors such that the Hamming distance between the actual variables (P) defined in Equation 1 is minimal. Recall that this would mean that the PUF output will not be able to detect the injected error. If the size of the checksum for each key row is a single bit, the error detection probability for an injected error would be 0.5 as the PUF can only provide an output of $\{0, 1\}$. In this case, the success rate for the attacker is considerably high. This is why we utilize a permutation block as shown in Figure 4.

The permutation block will essentially permute each input row r_i by a pre-determined permutation ρ_j where $j = (1, \dots, t)$ and $t < n$. Consequently, $\rho_j(r_i)$ is fed to the PUF in order to generate the (i, j) bit of the checksum. In short, for the secret key array S with size $k \times n$ and rows r_i , we calculate the (i, j) bit of the checksum S_w as

$$S_w(i, j) = \text{PUF}_Y(\rho_j(r_i)) \quad (3)$$

where the ρ_j 's are random permutations pre-chosen secretly and S_w is of size $k \times t$.

When this model is applied to secure the cryptographic devices against memory overwrite attacks, the robustness and security measure of the error detection scheme becomes a direct function of t , the number of the permutations used for each row. The probability of an error being detected is essentially the probability of an error changing the PUF output. However, we have seen in the previous section that the PUF output will sometimes be metastable. Therefore, we will again define an acceptable level of errors which will be a property of the system and which will not raise an alarm. Similar to the previous section we define this level as L . Now we can define the event for an error being detected. In particular, an error injected to row r_i will be detected provided that the following equation will not hold for more than L of the row's t checking bits.

$$\text{PUF}_Y(\rho_j(r_i)) = \text{PUF}_Y(\rho_j(r_i + e)) \quad (4)$$

where e indicates an error injected to the i^{th} row, e.g. bit flip of some memory cells. We now calculate this probability. Equation 4 is essentially the probability calculated in Equation 2. Therefore, we will again have to refer back to the Hamming distance between the ideal and the attacked PUF inputs. Because the permutations ρ_j are taken over all possible permutations, the attacker cannot control the effective location of his injected errors. To simplify the calculation we make the following assumption.

Assumption 1: *We assume an attacker model where the number of faults injected by the attacker is uniform over all possible number of faults.*

With Assumption 1 we can calculate the expected value of the Hamming distance between the P values of the original data and the faulty data when taken over all permutations to be equal to $n/2$. Going back to Equation 2 for

this particular Hamming distance the probability for Equation 3 to hold will be 0.5. With this, the detection probability of an error becomes $1 - 2^{-(t-L)}$.

At this point, it is important to note the trade-off between the area overhead and security level of the suggested mechanism. As the number of permutations t for each row increases, the security of the device gets stronger because the error detection probability increases. However, the area overhead also increases linearly with t due to the checksum storage space. The optimal value for t is an application dependent issue and can be adjusted according to the required security level or allowed area overhead.

Note that one can use error correcting codes to address the integrity issue. However, such a solution would require substantially more hardware for decoding the code words. Moreover, the PUF circuit has built-in fault resilience due to its sensitive characteristics. Consequently, any kind of fault injection or perturbation of the hardware will modify the result of the PUFs. This brings an additional level of security to the proposed key integrity and protection scheme. In addition, the solution we present here views the separate memory rows as independent entities. It is an interesting problem to explore combinations of the rows and columns, which might improve the error detection probability. Finally, the model here assumes that the checksum is hidden secretly. If a designer wishes to relax this condition different PUF designs should then be explored.

6 Error Detection Network Security

Concurrent error detection (CED) is one of the most common solutions against active fault attacks. The basic idea in these schemes is to calculate the expected result using predictor circuits in parallel to the main hardware branch, and compare if the predicted value of the result matches the actual value calculated in the main branch. A good overview along with elaborate examples about this mechanism can be found in [11]. An error signal is issued when these two paths do not produce agreeing results. This comparison along with the error signal generation are conducted by the error detection network (EDN) modules.

While the attackers are always assumed to target the main or predictor branch of a cryptographic device, the EDN network which is in fact the weakest link in the design is always assumed to be completely fault resilient. An attacker can deactivate the EDN by preventing an error signal from being issued or by just simply attacking the inputs of the EDN. Therefore, totally disabling/bypassing the error detection mechanism. To tackle this problem, we propose utilizing PUF structures to design secure and fault tolerant EDN blocks.

The suggested PUF based EDN mechanism is shown in Figure 5. Basically, the results coming from the main and predictor branches of the computation are first XOR-ed together. In the absence of an injected fault, the result will be the zero vector. The circuit starts by producing a fingerprint of the PUF response to an all zero bit challenge right before the circuit is deployed. This fingerprint is stored as the checksum bit. Throughout the circuit's operation, the XOR-ed results are continuously permuted and fed into the PUF circuit. This

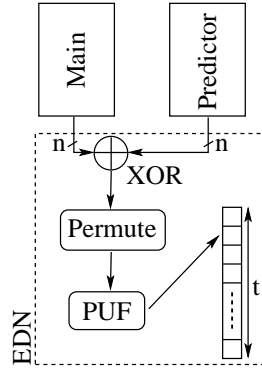


Fig. 5. PUF based EDN

permutation block implements the same functionality as in Section 5. In the absence of an error, the permutation will have no effect on the all zero vector, Therefore the output should always match the stored checksum bit. However, when an error is injected the output of the XOR will not be the all zero vector. This will cause the permutations to generate different challenge vectors which will consequently produce PUF outputs which are different from the checksum bit.

When the circuit detects a mismatch between the output of the PUF and the checksum bit an injected fault is assumed and an error signal can be issued. Similar to the analysis conducted in the previous two sections, the error detection capability of the EDN is dependent upon the number of applied permutations t , and can be formulated as $1 - 2^{-(t-L)}$. As in the previous sections, L here is the acceptable threshold of errors in the PUF response. The trade-off between the security and area overhead discussed in Section 5 also exists in this PUF based EDN methodology too.

Note that any attempt by the attacker to modify the voltage levels of the wires located inside the EDN will affect and change the result of the PUF due to its high sensitivity. This intrinsic tamper resistance of the PUF circuit acts as assurance against fault attacks targeting the EDN.

7 Conclusion

In this paper, we explored the integration of PUFs into the building blocks of finite state machines to provide security. In particular, we addressed the security of state-transitions (next-state logic) against fault-injection attacks, the integrity of secret information, and finally fault-resilience in error detection networks. We proposed PUF-based architectures for the security of these modules in a control unit, and showed that the probability of error detection is high. More importantly, the solution we propose provides security on the physical level as well as the logical level. Even if the adversary can find the appropriate fault to

inject, there will still be a good chance of being detected by the change in the PUF behavior. The designs we propose in this paper are described from a higher level and therefore are far from being final solutions ready for implementation. Rather, these solutions are mainly intended to open a new door for research in the area of unclonable protection of FSMs. Such mechanisms can provide strong error detection with a relatively low hardware overhead. Our work here is a first step in this direction.

References

1. S. Agmon. The relaxation method for linear inequalities. In *Canadian J. of Mathematics*, pages 382–392, 1964.
2. Ross Anderson and Markus Kuhn. Tamper resistance: a cautionary note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association.
3. Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 125–136, London, UK, 1998. Springer-Verlag.
4. Bar-El H. and Choukri H. and Naccache D. and Tunstall. M. and Whelan C. The sorcerer's apprentice guide to fault attacks. In *Proceedings of the IEEE*, volume 94, pages 370–382, February 2006.
5. M. Berg. Fault tolerant design techniques for asynchronous single event upsets within synchronous finite state machine architectures. In *7th International Military and Aerospace Programmable Logic Devices (MAPLD) Conference*. NASA, Sep 2004.
6. Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Transactions on Computers*, 52(4):492–505, 2003.
7. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 513–525, London, UK, 1997. Springer-Verlag.
8. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology - EuroCrypt'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Heidelberg, 1997. Springer. Proceedings.
9. P. Cunningham and R. Anderson and R. Mullins and G. Taylor and S. Moore. Improving Smart Card Security Using Self-Timed Circuits. In *Proceedings of the 8th international Symposium on Asynchronous Circuits and Systems* (April 08 - 11, 2002). ASYNC. IEEE Computer Society, Washington, DC, 211.
10. B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Delay-based Circuit Authentication and Applications. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 294–301, 2003.
11. G. Gaubatz, B. Sunar, and M.G. Karpovsky. Non-linear Residue Codes for Robust Public-Key Arithmetic. *Fault Diagnosis and Tolerance in Cryptography*, pages 173–184, 2006.

12. Gunnar Gaubatz and Berk Sunar. Robust finite field arithmetic for fault-tolerant public-key cryptography. In Luca Breveglieri and Israel Koren, editors, *2nd Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTTC 2005*, September 2005.
13. Gunnar Gaubatz, Berk Sunar, and Erkey Savas. Sequential Circuit Design for Embedded Cryptographic Applications Resilient to Adversarial Faults. *IEEE Transactions on Computers*, 57(1):126–138, 2008.
14. J. Guajardo, S.S. Kumar, G.J. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. *Cryptographic Hardware and Embedded Systems CHES 2007*.
15. Ghaith Hammouri, Erdinc Ozturk, and Berk Sunar. A Tamper-Proof, Lightweight and Secure Authentication Scheme. under review.
16. Ghaith Hammouri and Berk Sunar. PUF-HB: A Tamper-Resilient HB based Authentication Protocol. In *to appear in Proceedings of the Applied Cryptography and Network Security Conference – ACNS08*, 2008.
17. M. Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. *LECTURE NOTES IN COMPUTER SCIENCE*, 4727:135, 2007.
18. M. Joye and S.M. Yen. The Montgomery Powering Ladder. *Cryptographic Hardware and Embedded Systems-Ches 2002: 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002: Revised Papers*, 2002.
19. Mark Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Differential fault analysis attack resistant architectures for the advanced encryption standard. In *Proc. World Computing Congress*, 2004.
20. Mark Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, page 93, Washington, DC, USA, 2004. IEEE Computer Society.
21. Mark Karpovsky and Alexander Taubin. A new class of nonlinear systematic error detecting codes. *IEEE Trans Info Theory*, 50(8):1818–1820, 2004.
22. Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 21(12):1509–1517, 2002.
23. Osman Kocar. Estimation of keys stored in cmos cryptographic device after baking by using the charge shift. *Cryptology ePrint Archive*, Report 2007/134, 2007. <http://eprint.iacr.org/>.
24. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. *Advances in Cryptology-Crypto'99: 19th Annual International Cryptology Conference, Santa Barbara, California, USA August 15-19, 1999 Proceedings*, 1999.
25. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, London, UK, 1996. Springer-Verlag.
26. Andrzej Krasniewski. Concurrent error detection in sequential circuits implemented using fpgas with embedded memory blocks. In *Proceedings of the 10th IEEE International On-Line Testing Symposium (IOLTS'04)*, 2004.
27. Konrad J. Kulikowski, Mark Karpovsky, and Alexander Taubin. Robust codes for fault attack resistant cryptographic hardware. In *Workshop on Fault Diagnosis and Tolerance in Cryptography 2005 (FTDC'05)*, 2005.

28. Konrad J. Kulikowski, Mark Karpovsky, and Alexander Taubin. Fault attack resistant cryptographic hardware with uniform error detection. In *Lecture Notes in Computer Science*, pages 185–195. Springer Berlin / Heidelberg, 2006.
29. J. W. Lee, L. Daihyun, B. Gassend, G. E. Suh and M. van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *Symposium of VLSI Circuits*, pages 176–179, 2004.
30. Daihyun Lim, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srinivas Devadas. Extracting secret keys from integrated circuits. *IEEE Trans. VLSI Syst.*, 13(10):1200–1205, 2005.
31. David Naccache. Finding faults. *IEEE Security and Privacy*, 3(5):61–65, 2005.
32. Erdinc Ozturk, Ghaith Hammouri, and Berk Sunar. Towards robust low cost authentication for pervasive devices. In *PERCOM '08: Proceedings of the Sixth IEEE International Conference on Pervasive Computing and Communications*, 2008.
33. R. Posch. Protecting Devices by Active Coating. *Journal of Universal Computer Science*, 4(7):652–668, 1998.
34. P.S. Ravikanth. *Physical One-Way Functions*. PhD thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2001.
35. Cornelis Roos, Tamas Terlaky, and Jean-Philippe Vial. *Interior Point Methods for Linear Optimization*. Springer, second edition, 2005.
36. J.M. Schmidt and M. Hutter. Optical and em fault-attacks on crt-based rsa: Concrete results. In *Austrochip '07: Proceedings of the 15th Austrian Workshop on Microelectronics*, 2007.
37. B. Skoric, S. Maubach, T. Kevenaer, and P. Tuyls. Information-theoretic Analysis of Coating PUFs. Cryptology ePrint Archive, Report 2006/101, 2006.
38. S.P. Skorobogatov and R.J. Anderson. Optical Fault Induction Attacks. *Cryptographic Hardware and Embedded Systems-Ches 2002: 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002: Revised Papers*, 2002.
39. D. Sokolov and J. Murphy and A.V. Bystrov and A. Yakovlev Design and Analysis of Dual-Rail Circuits for Security Applications. *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 449-460, April, 2005.
40. P. Tuyls, G.J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters. Read-Proof Hardware from Protective Coatings. *Cryptographic Hardware and Embedded SystemsCHES*, pages 10–13, 2006.
41. P. Tuyls and B. Skoric. Secret Key Generation from Classical Physics: Physical Uncloneable Functions. In S. Mukherjee, E. Aarts, R. Roovers, F. Widdershoven, and M. Ouwerkerk, editors, *AmIware: Hardware Technology Drivers of Ambient Intelligence*, volume 5 of *Philips Research Book Series*. Springer-Verlag, Sep 2006.
42. J. Waddle and D. Wagner. Fault Attacks on Dual-Rail Encoded Systems. In *Proceedings of the 21st Annual Computer Security Applications Conference* (December 05 - 09, 2005). ACSAC. IEEE Computer Society, Washington, DC, 483-494. DOI=<http://dx.doi.org/10.1109/CSAC.2005.25>