A Design Method for Remote Integrity Checking of Complex PCBs

Aydin Aysu, Shravya Gaddam, Harsha Mandadi, Carol Pinto, Luke Wegryn, Patrick Schaumont
Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, USA
{aydinay,shravyag,harsham,carol89,lwegryn,schaum}@vt.edu

***Abstract*—Modern, complex printed circuit boards contain high-end commercial off-the-shelf components such as high-capacity FPGAs and expensive peripherals. This paper describes a strategy to build a hardware attestation protocol for such a board. The owner or operator of the PCB wants to achieve the assurance that the board installed in the field is physically the same as the one that was originally deployed. Our methodology builds a unique identifier for the PCB by cryptographically linking individual component-level identifiers from the board. The component-level identifiers are implemented using Physical Unclonable Functions (PUF) within the components of the board. We discuss a generic methodology for design and dimensioning of the critical post-processing parameters of the PUF, and we present several strategies to combine multiple PUF into a combined Fusion PUF. We present a prototype of the proposed technique on an FPGA board running μClinux, and we characterize its performance on a population of 22 PCBs.**

***Index Terms*—Physical Unclonable Functions; Integrity Verification; PCB.**

I. Introduction

Network routers and deep-packet inspection engines rely on high-end hardware installed deeply inside of a network. Such remote, unattended hardware represents a significant investment, and it is desirable to verify its presence after installation. This can reveal hardware tampering along the supply chain from manufacturing to remote installation. This paper presents a solution for hardware attestation of a printed circuit board (PCB) that includes a collection of expensive and complex electronic components.

A naive solution to the problem is to add a board-level identifier, such as a hard-coded identifier in non-volatile memory. Of course, an unprotected non-volatile memory is not tamper-proof, and it cannot be used for hardware attestation. But even a tamper-protected non-volatile memory is not an adequate solution. A protocol based on authenticating this remote hard-coded identifier would only demonstrate the presence of the non-volatile memory. It would still require a tamper-resistant integration of the physical PCB with the non-volatile memory.

We propose a mechanism that authenticates all major computational and memory elements of the PCB itself as part of attesting the PCB. Such elements can include Field Programmable Gate Arrays (FPGAs), SRAM memory, processors, as well as non-volatile memory. By extracting a hardware fingerprint from these components using Physical Unclonable Functions (PUFs), we can construct the fingerprint of an entire PCB by considering the *ensemble* of chips on the PCB.

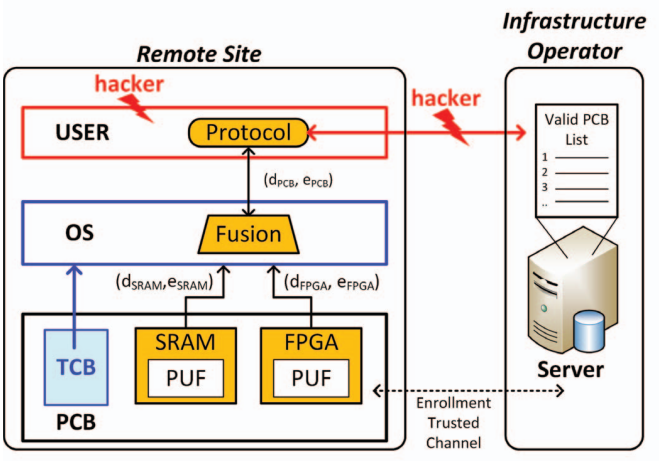


Figure 1. Hardware Attestation using Fusion PUF

A. Hardware Attestation with Fusion PUF

Figure 1 sets up the problem that we aim to solve. A PCB is installed at a remote site, and it needs to be attested by an infrastructure operator. Each PCB has a unique identity, and the infrastructure operator maintains a list of installed, valid PCBs that is used to verify if a tested PCB is valid.

The PCB runs a remote attestation protocol that demonstrates the presence of the PCB using a PCB-level PUF, constructed from individual component level PUFs. We will use the term *Fusion PUF* to describe this PCB-level PUF. Each individual component PUF is characterized through two parameters (d, e), including the probability of a bit error e, and the probability d of a bit-flip between two instances of the same component. Both e and d are fractional numbers; over a population of components each configured with n-bit PUF, we would expect an intra-Hamming distance of $n.e$ and an inter-Hamming distance of $n.d$.

The rationale for using a PUF is to ensure that a proof of identity can be established on the presence of components on the PCB, rather than on a stored secret. This is driven by similar concerns as a two-factor authentication protocol - we transform the authentication problem from *something that the PCB stores* to *something that the PCB has*. We aim to establish this assurance with a security level of 80-bit and a probability of error of one part in one million.

B. Threat Model and Tamper Resistance

The infrastructure operator runs remote hardware attestation on the PCB, and wants to correctly identify every previously installed and enrolled PCB. In addition, the operator also wants

978-3-9815370-6-2/DATE16/

to systematically detect *unknown*, previously non-enrolled PCB's. The cloning of a single PCB identifier has a limited risk, since the server can flag the systematic reuse of a single identifier (eg. in a series of cloned and hacked PCB's).

We assume that the PUF enrollment process is secure, and that the database with PCB identities is secure. An adversary may tamper with the communications between the PCB and the infrastructure operator, and an adversary may also tamper with the application-level software running on the PCB. However, we assume that the PCB has a minimal trusted computing base (TCB) to enforce privileged access to the PUF output. Hence, the OS, or at least the part of the OS that computes the Fusion PUF, is protected through the TCB. Recent research results have demonstrated that such a TCB can be obtained on constrained hardware [1]–[3]. At the level of the hardware, we assume that the TCB is tamper-resistant, but not necessarily the other components like FPGA or SRAM. Low-level hardware tampering, fault attacks, and side-channel attacks are out-of-scope for this research.

C. Previous Work in PCB Anti-Tamper

PCB's have been previously evaluated as a potential target for Hardware Trojans [4]. A line of defense against such attacks is to protect the PCB itself. Wei presents a solution based on capacitor integration in intra-board PCB layers [5]. A similar (commercial) solution is a unique tag physically integrated into the board. Both of these solutions require dedicated PCB technology, in contrast to a technology that would directly authenticate the components on the PCB. Zheng presents ScanPUF, which extracts chip identifiers based on their scan chain [6]. This technique is component independent, but requires the board test infrastructure to remain enabled after PCB production and deployment. In contrast, our technique uses component-intrinsic PUFs.

D. Contributions and Outline of the Paper

This paper brings three contributions. First, the paper describes a methodology for dimensioning and parameter design of PUFs. Second, we propose an efficient integration of multiple PUF into a Fusion PUF. Third, we demonstrate a prototype and validate it on a collection of 22 board.

In the following Section, we introduce a methodology for PUF protocols. In Section III, we describe the Fusion PUF and its expected performance. In Section IV we present implementation details of our prototype, and in Section V we analyze the cost, complexity, and measured performance. Section VI concludes the paper.

II. Parameter Definition in PUF protocols

We consider the coding requirements for the use of a PUF-generated key as the basis for a classic MAC-based authentication protocol. Such a protocol works by enrolling the PUF key once, and by verifying it later through a MAC-based challenge-response. Compared to a traditional implementation, the secret key is noisy and thus a protocol that uses it requires additional analysis. This analysis reveals how many PUF response bits are required to generate a stable key with a given target entropy. The analysis also reveals how much error coding is required to obtain a key with a given target error rate.

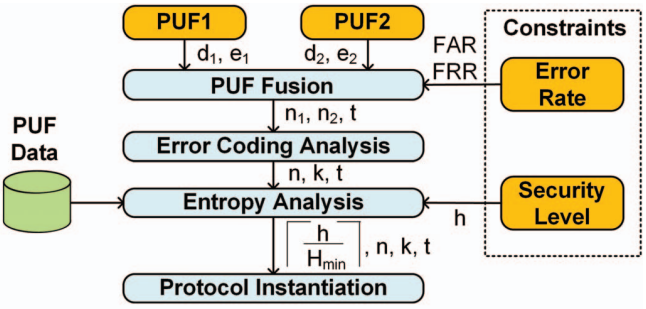


Figure 2. The Design Flow of PUF-based Protocols

Figure 2 shows a sequence of steps to derive parameters of a PUF-based authentication protocol. We assume two PUF components PUF_1 and PUF_2 with bit error probabilities e_1 and e_2, and inter-device bit-flip probabilities of d_1 and d_2. These probabilities can be estimated for the population by statistical means [7]. Additional constraints include the desired misidentification rates including the false accept rate (FAR) and the false reject rate (FRR), and desired security level (h).

- **PUF Fusion** defines how to merge PUF components with different noise (e_1, e_2) and inter-distance (d_1,d_2) characteristics, and how to derive the required PUF bits n_1, n_2, with the identification threshold t that satisfies the FAR and FRR. The choice of a pair n_1, n_2 that satisfies the FAR and FRR may not be unique. In that case, the designer should consider multiple pairs as possible candidates for the following steps. The most suitable pair would be the one that meets an optimality criterion for the entire system.
- Next, **Error Coding** searches a suitable coding scheme that can correct up to t-bit errors on an $n = n_1 + n_2$ bit codeword. Using this coding scheme with code-offset mechanism [8], it is possible to generate a helper data to remove the noise of genuine PUF responses. However, this helper data contains parity information which reduces the entropy content of the PUF response.
- **Entropy Analysis** calculates the original entropy content of the PUFs and computes a lower-bound H_{min} on the residual entropy after the error coding. To accumulate the target entropy h, the protocol can process $\lceil h/H_{min} \rceil$ blocks of n-bit PUF responses. To satisfy the FAR and FRR requirements, the protocol has to apply an error correction of up to t-bits on each of these n-bit responses.

III. PUF Fusion

In this section, we investigate three alternatives to merge PUF components with different error (e) and inter-distance (d) characteristics: protocol-level fusion, hardware-level fusion with concatenation, and hardware-level fusion with XOR operation. Figure 3 visualizes these three methods.

A. Protocol-level fusion

In protocol-level fusion (Figure 3 (a)), the device extracts PUF responses and separately generates helper data of individual components. Likewise, PUF responses are reconstructed

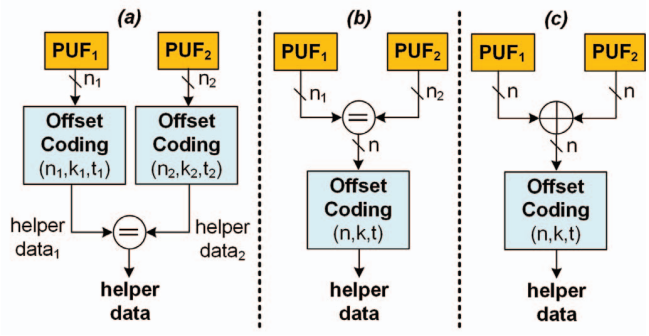


Figure 3. Representation of the three fusion methods

by separate error decoding functions with distinct parameters. Then, the verifier could detect if any of the PUF components are fake. Therefore, on an authentic PCB, the FRR and FAR requirements has to individually hold for every component. The authentication protocol sets an identification threshold t and assumes that an enrolled n-bit PUF response could have at most t-bit noise. Hence, given the probability of bit error e we can calculate the FRR by using Equation 1. The FRR corresponds to the likelihood of observing more than t-bit errors on an n-bit response from the same PUF instance. Likewise, given the probability d of a bit-flip between two instances of the same component, we can calculate the probability of FAR by using Equation 2. The FAR corresponds to the likelihood of observing less than a t-bit distance between n-bit responses from different PUF instances. These equations estimate error rates by using the cumulative binomial distribution function $F_c(t, n, p)$. This function computes the probability of up to t successes in n independent trials each having a success rate of p. Since the error coding capability is proportional to its entropy leakage, it is desirable to set t as the minimum integer value that satisfies FAR and FRR.

$$FRR(t, n) = 1 - F_c(t, n, e) \tag{1}$$

$$FAR(t, n) = F_c(t, n, d) \tag{2}$$

The disadvantage of protocol level fusion is that two separate block codes have to run on the device and on the server. This may limit its usability in an embedded context, where the program memory of the target micro-controller or the logic area of the target FPGA is limited. If the system is designed for a single block code, the error-coding parameters must satisfy the error rates of the lowest quality PUF. This would cause performing excessive error-corrections on high-quality PUFs, and getting less entropy from them.

B. Hardware-level fusion

In hardware level-fusion, the device extracts the PUF response on individual components, combines them, and generates a single helper data for the entire collection. Now, the kernel perceives PUF block as a unified single instance which we call the Fusion PUF. The system applies a single block code on the Fusion PUF and generates a single helper data. Using this helper data, the verifier should be able to authenticate the PCB with the target error rates. Depending on how PUF fusion combines PUF responses, the FRR and FAR are calculated differently.

1) Concatenation-based: In concatenation-based fusion (Figure 3 (b)), the Fusion PUF concatenates the response PUF_1 and PUF_2 respectively having n_1 bits and n_2 bits. Hence, the response size of Fusion PUF is $n = n_1 + n_2$ bits. On average, the Fusion PUF with n-bit response would have $n_1e_1+n_2e_2$-bit noise. If this noise is more than t for a genuine PUF, then this would lead to false reject errors. Equation 3 shows how to calculate this probability for parameters n_1, n_2, and t. $F_p(t, n, p)$ is the binomial probability density function which computes the probability of exactly t successes in n independent trials each having a success rate of p. For a successful authentication, if one component of a genuine Fusion PUF has exactly $t-i$-bit noise, the other must at most has i-bit noise, and vice versa. The sum of all such error combinations from 0 to t would give the total probability of observing errors with threshold t. Although it achieves the same FRR, this scheme would allow correcting some error occurrences that are not possible with the protocol-level fusion. For example, if PUF_1 has $t_1 + 1$-bit error, fusing at protocol-level would always fail, but the hardware-level fusion would succeed if PUF_2 has less than $t - t_1$-bit error.

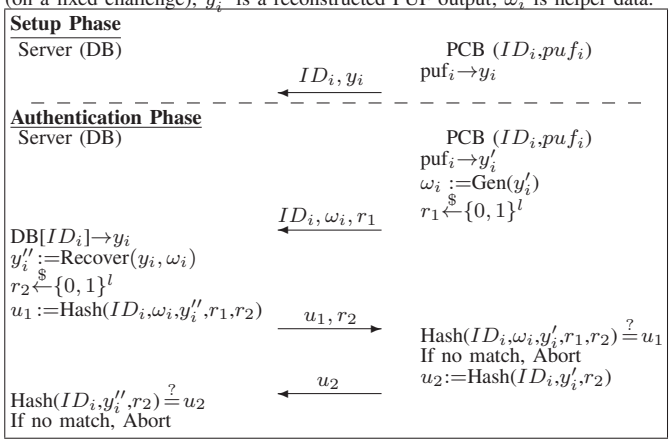
$$\begin{aligned} FRR_1(n_1, n_2, t) &= \sum_{i=0}^{t} (F_p(i, n_2, e_2)(1\text{-}F_c(t\text{-}i, n_1, e_1))) \\ FRR_2(n_1, n_2, t) &= \sum_{i=0}^{t} (F_p(i, n_1, e_1)(1\text{-}F_c(t\text{-}i, n_2, e_2))) \\ FRR &= max\{FRR_1, FRR_2\} \end{aligned} \tag{3}$$

$$\begin{aligned} FAR_1(n_1, n_2, t) &= \sum_{i=0}^{t} (F_p(i, n_2, e_2)F_c(t\text{-}i, n_1, d_1)) \\ FAR_2(n_1, n_2, t) &= \sum_{i=0}^{t} (F_p(i, n_1, e_1)F_c(t\text{-}i, n_2, d_2)) \\ FAR &= max\{FAR_1, FAR_2\} \end{aligned} \tag{4}$$

An adversary can try to trick the protocol by only changing one Fusion PUF component. If such a PCB is validated, it would be a false accept. Therefore, FAR should reflect the necessity to detect even if a single PUF component is changed within the PCB. In this scenario, the intra-Hamming distance of the genuine PUF plus the inter-Hamming distance of the fake PUF should be greater than the threshold t. For example, if PUF_1 is fake, and if PUF_2 has i-bit error, the Fusion PUF will be misidentified as valid (FAR) if PUF_1 have at most $t-i$ bit inter-distance. The sum of all such combinations of i from 0 to t would give the total FAR. This would also guarantee an invalid authentication if both PUF components are fake. Equation 4 formulates these conditions for the two PUF setting and for parameters n_1, n_2, and t.

2) XOR-based: XOR-based fusion (Figure 3 (c)) uses an XOR operator to mix the PUF responses of equal size n. The parameters for threshold t and the size n can be calculated exactly like a single PUF instance by using equations 1 and 2.

Figure 4. The Reverse Fuzzy Extraction protocol on a PCB with a public identity ID_i [9]. r_1 and r_2 are true random numbers; y_i, y_i' are PUF outputs (on a fixed challenge); y_i'' is a reconstructed PUF output; ω_i is helper data.



However, the binomial estimators e and d have to be modified as the XOR operation will affect the characteristics of the Fusion PUF. Now, a Fusion PUF bit is noisy if one of its input is noisy. Therefore, the bit error e_V of the Fusion PUF is defined as $e_1+e_2-(e_1e_2)$. When only one PUF component is fake, a Fusion PUF bit flips if one of its input is flipped. Assuming that either PUF can be fake, the d_V is defined as $\min\{d_1(1\text{-}e_2)+(1\text{-}d_1)e_2,\ d_2(1\text{-}e_1)+(1\text{-}d_2)e_1\}$. This would also guarantee detecting when both PUF components are fake.

IV. System Demonstrator

A. System Protocol

Our prototype follows the Reverse Fuzzy Extractor protocol [9] (described as *modified reverse FE protocol* in [10]) which provides mutual authentication between a server and a constrained device. Figure 4 shows the steps of this protocol. During enrollment, the PCB registers the response of the Fusion PUF y_i and a public identifier ID_i.

When the server initiates the authentication, PCB generates a response (y_i') and the associated helper data (ω_i) on this noisy response. Then it replies, with the public identifier, helper data, and a random nonce (r_1). The server fetches the enrolled response from its database by using the public identifier, and reconstructs the noisy PUF response (y_i) by using the helper data. After this step, the two authenticating parties can check each others knowledge on the noisy PUF response by hash functions with random nonces (r_1,r_2). The protocol is constructed as a server-first mutual authentication.

B. Software Architecture

Figure 5 shows the software architecture of our prototype. It consists of three components: (i) the μClinux operating system with kernel modules, (ii) the Client Application on the device that interacts with the verifier, and (iii) the Server Application used by the verifier to authenticate the PCB. These components implement the system protocol. We clarify that our system demonstrator does *not* include a Trusted Computing Base (TCB). The use of μClinux on a Nios-II was done to demonstrate the integration of the Fusion PUF as a combination of hardware and software, and to support easy integration with a networking stack.

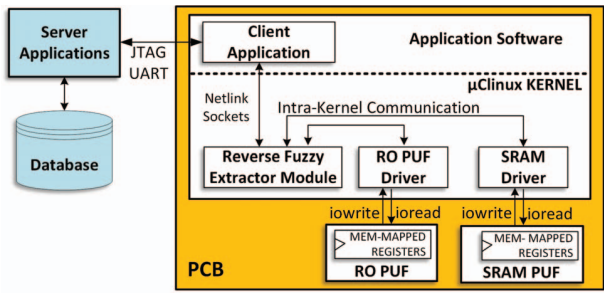


Figure 5. Block Diagram of Software Architecture

Table I. Data flow during protocol execution

Protocol Operation	Server App	Client App	FE Mod	RO PUF Driver	SRAM Driver
	Req				
$puf_i \rightarrow y_i'$				Read	Read
ω_i :=Gen(y_i')			Comp		
$r_1 \xleftarrow{\$} \{0,1\}^l$			Comp		
ID_i, ω_i, r_1		Send			
ID_i, ω_i, r_1	Recv				
DB[ID_i]$\rightarrow y_i$	Comp				
y_i'' :=Recover(y_i, ω_i)	Comp				
$r_2 \xleftarrow{\$} \{0,1\}^l$	Comp				
u_1 :=Hash(...)	Comp				
u_1, r_2	Send				
u_1, r_2		Recv			
Hash(...)			Comp		
$\stackrel{?}{=} u_1$		**Auth**			
u_2:=Hash(...)			Comp		
$u2$		Send			
$u2$	Recv				
Hash(...)	Comp				
$\stackrel{?}{=} u_2$	**Auth**				

The protocol needs a secure method to enroll the PUF response once. Afterwards, the PUF output is privileged information throughout its processing. In our prototype, PUF-related computing is entirely carried out within the μClinux kernel and no user level application has access to this data. We designed three kernel modules to implement the PUF post-processing and the sensitive steps of the authentication protocol. These are the Reverse Fuzzy Extractor Module (RFEM), the SRAM driver, and the RO PUF driver. Table I summarizes the operations of these modules.

The SRAM driver maps the SRAM into the μClinux kernel memory space which can be read by RFEM. RO PUF driver can access the RO frequencies through memory mapped registers and post-process it to generate PUF response. RFEM interacts with SRAM and RO PUF drivers to generate the PUF response. This response is then processed by RFEM to generate the helper data. RFEM is also responsible for generating the SHA-256 hashes (u_1, u_2) used in the verification protocol. The Client application runs on the μClinux operating system on the device. This entity is responsible for receiving and processing messages from the verifier. It receives the messages sent by the host application over JTAG UART and handles kernel communications through Netlink Sockets.

C. Hardware Architecture

1) System design: Figure 6 illustrates the hardware architecture of the prototype. Our design integrates a Nios-II softcore processor with SRAM, SDRAM, UART and several FPGA components (RO PUF, SRAM controller and SDRAM

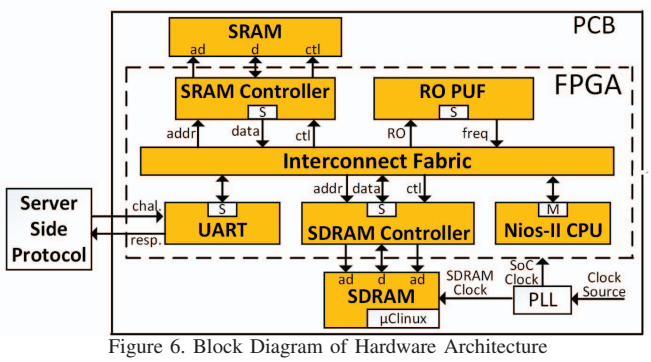


Figure 6. Block Diagram of Hardware Architecture

controller). The prototype does not include a dedicated Hardware True Random Number generator, but relies on the operating system to provide a source of true randomness. When the implementation is built on top of a TCB, a high-quality TRNG is typically included as well.

2) Hardware and Software Integration: To connect hardware components on FPGA, our design makes use of the Avalon Memory Mapped interface, an address based interface with master-slave connections. Nios-II processor (data and instruction master) controls the signals of data slave components (SRAM Controller, RO-PUF, SDRAM controller, JTAG) and of instruction slave components (SDRAM controller). The SRAM and RO-PUF Kernel modules communicate with the respective PUF modules through Memory Mapped registers. The Kernel module initiates the input arguments to the PUF via NIOS-II processor, by placing the arguments in the Memory Mapped registers. The hardware can access these registers and generates the PUF data based on input arguments. Generated results are placed back in the respective memory mapped registers which are retrieved by the respective kernel modules.

3) SRAM PUF Design: An SRAM PUF exploits the transistor mismatches of SRAM cells [11]. Due to the process manufacturing variation, each cell randomly has a preferred power-up state. In our system demonstrator, SRAM PUF design uses on-board static RAM (ISSI IS61WV102416BLL) and an FSM controller. The SRAM Controller is interfaced to the SoC as a slave, and it handles the communication between the NIOS-II processor and the 256K x 16 CMOS static RAM chip. The controller receives the SRAM memory address, reads power-up state of that location, and transmits the values to the processor through the Avalon Fabric.

4) RO-PUF Design: The RO-PUF exploits delay variations in the logic elements to produce a unique n-bit identifier [12], [13]. The design works by chaining an odd number of inverters and connecting the output of the last inverter to the input of the first inverter. This way, each RO produces an oscillating output with a frequency dependent on how quickly the looping signal propagates through the logic elements. The output (frequency) of an RO (A) is compared to the output of another RO (B), and either a 1 or a 0 is produced depending on whether A or B has a faster frequency.

We follow the guidelines of Feiten [14], which describes several optimizations on Altera FPGAs, including the optimal logic design for ROs and manual back-end tool manipulations. Then, we minimize systematic bias by comparing the ROs that have closest frequency, similar to sequential pairing [15]. The pairing is not unique to each individual FPGA but is based on the average of all observed frequencies on all FPGAs. Therefore, it can also be implemented in hardware to prevent helper data manipulation attacks [16]. This method would generate n-1 bit responses from n ROs. In our design, we use 256 ROs to generate a 255-bit PUF output. To read stabilized RO frequencies, the system waits for 100 ms before starting a measurement. Then, it counts the oscillations for 250 ms.

Table II. The Results of Parameter Extraction for d_1=0.4838, d_2=0.4867, e_1=0.0522, e_2=0.0128

Fusion mode	Protocol-lvl		XOR	CAT-`Balanced`
	SRAM (n_1,t_1)/ (n,k,t)	FPGA (n_2,t_2)/ (n,k,t)	(n_1,n_2,t)/ (n,k,t)	(n_1,n_2,t)/ (n,k,t)
Min. bit	(73,15)/ —	(48,7)/ (48,9,15)	(84,84,19)/ —	(80,74,18)/ (154,30,18)
$n_1+n_2=255$	(127,21)/ (127,29,21)	(127,10)/ (127,64,10)	(127,127,24)/ (127,15,27)	**(128,127,25)**/ **(255,91,25)**

V. Implementation Results

A. Parameter Extraction for Fusion PUF

We derive the binomial estimators from experiments on 22 boards. We implement a simple brute-force search to find all possible combinations of n_1 and n_2 up to 511 bits. Table II shows different configurations for Fusion PUF and associated parameters. *Min. bit* shows the combination of n_1, n_2 that uses the minimum number of PUF response bits n to achieve the FAR and FRR requirement. Due to the construction of BCH codes, the optimal error-coding efficiency occurs when the PUF size is equal to the order of the generator polynomial (ie. $n = 2^\omega - 1$). $n_1+n_2 = 255$ corresponds to this setting where the configurations use a total of 255-bit PUF response.

SRAM PUF noise limits the applicability of XOR-based fusion on our solution. However, it is useful if the system has robust but low-entropy PUF constructions. Concatenation (CAT), on the other hand, requires a single and simple error correction mechanism, and is on par with the protocol-level fusion method in terms of residual entropy. There are 76 possible combinations of n_1, n_2, and t that satisfy Equation 3, 4, and $n_1 + n_2 = 255$. We opted to implement a balanced concatenation combination that merges 128-bits of SRAM with 127-bits of RO PUF response. This selection is also preferred if one wants to evenly distribute the entropy among Fusion PUF input sources with similar min-entropy rates. The residual entropy (H_{min}) is approximately 40-bits for our observed min-entropy rate of 80% while it is 91-bits for an ideal entropy rate of 100%. To achieve the target entropy of 80-bits, the process should iterate ($\lceil h/H_{min} \rceil$) twice, and use a total of 256-bit SRAM response and 254-bit RO PUF response.

The parameters of our authentication protocol are a concatenation-based fusion with (n_1=128, n_2=127) concatenation, $\lceil h/H_{min} \rceil = 2$, with BCH error coding parameters $n_{BCH} = 255$, $k_{BCH} = 91$, and $t_{BCH} = 25$.

B. Implementation Cost

We mapped our system demonstrator on the Altera DE2-115 board. We implemented the device-side hardware components using the Altera Quartus II tool chain version 15.0. The SoC integration of the composing elements is handled through the QSYS system integration tool. Table III shows the hardware utilization details of FPGA components. The results show that

Table III. Hardware Utilization for FPGA Components

Module	LE	BRAM	DSP
Nios-II	2254	8	4
RO-PUF	4385	-	-
SRAM controller	5	-	-
SDRAM controller	327	-	-
JTAG-UART	146	1	-
PLL	8	-	-
Total	7125	9	4

Table IV. Memory Requirements of Software Components

Module	Codesize (Bytes)	Ratio (%)
RO-PUF	38,820	2.28
SRAM PUF	35,128	2.07
Fuzzy Extractor	116,680	6.86
Board Application	35,360	2.08
μClinux Kernel	1,475,069	86.7
Total	1,701,057	100

the RO-PUF design with 256 ROs and its controller consumes most of the FPGA resources. Table IV lists the memory requirement of software modules. μClinux occupies the majority of the memory and the proposed protocol operations brings a total memory overhead of 13.3%.

C. Performance Evaluation

There are two major steps in the PCB-side of authentication protocol. Step I is to acknowledge an authentication request, to generate the PUF response and the associated helper data (ω_i), to compute the one-time random nonce (r_1), and to send it to the server with the public ID (ID_i). Step II is to check the correctness of the authentication server's response (u_1), to compute a hash over the identification attributes (u_2), and to send it to the server for verification. Table V lists the round trip communication time and computation time required for each step in the protocol implementation. Table VI shows the breakdown of the PCB-side operations.

VI. Conclusions

In this paper we introduced a mechanism to authenticate a PCB by cryptographically linking identifiers from lower level components. The technique enables to extend the trust obtained from a trusted computing base to the physical collection of components on the PCB. We believe that PUF technology is a viable, common-sense solution for this and many other problems that involve physical authentication of electronics. In other words, rather than building a strongly tamper-resistant PCB technology, we can limit physical tamper resistance to the TCB (trusted computing base) and extend its capabilities to other components by configuring PUFs in them. We are currently extending this basic concept to a comprehensive set of PUF technologies, and we are developing a method for secure enrollement of the Fusion PUF.

Acknowledgements

This research is supported in part by CISCO Systems Inc, and by the National Science Foundation grant no 1314598.

Table V. Execution time of Protocol Operations

Round trip	Payload (Bytes)	Time (s)
PCB-side Step 1	1024	191.29
PCB-side Step 2	1024	9.35
Total Time of Protocol	1024	200.71

Table VI. Breakdown of Protocol Operation

Authentication Step	Response Time	Unit
SRAM Read	249	μs
RO Read	178.19	s
Overall PUF Generation	178.21	s
BCH Encoding	37.39	ms
Random Number Generator (91 bits)	12.17	ms
u_1 Hash	2.15	ms
u_2 Hash	0.28	ms
Overall PUF processing	178.31	s

References

[1] J. Winter, "Trusted computing building blocks for embedded linux-based ARM trustzone platforms," in *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC, Alexandria, VA, USA, October 31*, 2008, pp. 21–30.

[2] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. V. Herrewege *et al.*, "Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base," in *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16*, 2013, pp. 479–494.

[3] F. F. Brasser, B. E. Mahjoub, A. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: tiny trust anchor for tiny devices," in *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11*, 2015, pp. 34:1–34:6.

[4] S. Ghosh, A. Basak, and S. Bhunia, "How Secure Are Printed Circuit Boards Against Trojan Attacks?" *IEEE Design & Test*, vol. 32, no. 2, pp. 7–16, 2015.

[5] L. Wei, C. Song, Y. Liu, J. Zhang, F. Yuan *et al.*, "BoardPUF: Physical Unclonable Functions for Printed Circuit Board Authentication," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD, Austin, USA, November 2-6*, 2015, pp. 152–158.

[6] Y. Zheng, A. R. Krishna, and S. Bhunia, "ScanPUF: Robust ultralow-overhead PUF using scan chain," in *18th Asia and South Pacific Design Automation Conference, ASP-DAC 2013, Yokohama, Japan, January 22-25*, 2013, pp. 626–631.

[7] R. Maes, "Implementation and Experimental Analysis of Intrinsic PUFs," in *Physically Unclonable Functions*. Springer Berlin Heidelberg, 2013, pp. 81–115.

[8] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," *SIAM Journal on Computing*, vol. 38, no. 1, pp. 97–139, 2008.

[9] R. Maes, "PUF-Based Entity Identification and Authentication," in *Physically Unclonable Functions*. Springer Berlin Heidelberg, 2013, pp. 117–141.

[10] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede, "A Survey on Lightweight Entity Authentication with Strong PUFs," *ACM Comput. Surv.*, vol. 48, no. 2, pp. 26:1–26:42, Oct. 2015.

[11] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proceedings of the 44th Annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 9–14.

[12] D. E. Holcomb, W. P. Burleson, and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proceedings of the Conference on RFID Security*, vol. 7, 2007.

[13] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Cryptographic Hardware and Embedded Systems - CHES*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds. Springer Berlin Heidelberg, 2007, vol. 4727, pp. 63–80.

[14] L. Feiten, A. Spilla, M. Sauer, T. Schubert, and B. Becker, "Analysis of Ring Oscillator PUFs on 60nm FPGAs," *European cooperation in science and technology*.

[15] C.-E. Yin and G. Qu, "LISA: Maximizing RO PUF's secret extraction," in *Hardware-Oriented Security and Trust (HOST),IEEE International Symposium on*, June 2010, pp. 100–105.

[16] J. Delvaux and I. Verbauwhede, "Key-recovery Attacks on Various RO PUF Constructions via Helper Data Manipulation," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 72:1–72:6.