PUFatt: Embedded Platform Attestation Based on Novel Processor-Based PUFs

Joonho Kong[1], Farinaz Koushanfar[1], Praveen K. Pendyala[2],
Ahmad-Reza Sadeghi[3], and Christian Wachsmann[4]

[1]Dept. of ECE, Rice University
[2]Indian Institute of Technology, Mumbai
[3]Technische Universität Darmstadt
[4]Intel CRI for Secure Computing at TU Darmstadt

{joonho.kong, farinaz}@rice.edu, praveendath92@iitb.ac.in,
{ahmad.sadeghi, christian.wachsmann}@trust.cased.de

ABSTRACT

Software-based attestation schemes aim at proving the integrity of code and data residing on a platform to a verifying party. However, they do not bind the hardware characteristics to the attestation protocol and are vulnerable to impersonation attacks.

We present PUFatt, a new automatable method for linking software-based attestation to intrinsic device characteristics by means of a novel *processor-based* Physically Unclonable Function, which enables secure timed (and even) remote attestation particularly suitable for embedded and low-cost devices. Our proof-of-concept implementation on FPGA demonstrates the effectiveness, applicability and practicability of the approach.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Security

Keywords

Attestation, Physically Unclonable Function (PUF)

1. INTRODUCTION

As embedded and mobile systems are increasingly permeating our information ecosystem, they are also being progressively utilized in security and safety-critical applications. This generates an increasing need for enabling technologies to validate and verify the integrity of a system's software state against malicious code (*attestation*). For this purpose, various approaches to attestation have been proposed [26]. Common to all of them is that the platform to be evaluated (*prover*) sends a status report of its current configuration to another platform (*verifier*) to demonstrate that it is in a known and thus trustworthy state. Current attestation techniques can be viewed as a continuum ranging from fully hardware-supported attestation using secure coprocessors [24, 36] to attestation schemes requiring no explicit hardware support as in software-based attestation [32, 31, 6, 10, 16, 15]. The solutions based on security hardware modules (such as the TPM [36]) are inappropriate for resource-constrained embedded systems, while purely software-based attestation relies on strong assumptions, such as tamper-evident hardware and out-of-band (e.g., visual) prover authentication, which are hard to achieve in practice.

A practical lightweight attestation scheme for embedded devices should have low hardware overhead and reasonable attestation times. Software-based attestation follows this paradigm since it does not require any cryptographic secrets or security hardware. However, software attestation cannot explicitly authenticate the underlying hardware components, making it vulnerable to impersonation attacks [30]. To address this problem *timed attestation* protocols have been proposed that bind the software-attestation algorithm to the underlying hardware, e.g., by exploiting the side-effects of CPU operation [12] or by using security features in hardware [29, 15, 10]. However, these approaches have been shown to be ineffective [33] or they require secure hardware components that are too complex or expensive for resource-constrained embedded systems. In this context, Physically Unclonable Functions (PUFs) that generate a unique hardware-specific output (*response*) to each input (*challenge*), are a promising technology to bind software-based attestation schemes to a particular physical hardware platform and enable lightweight remote attestation particularly suitable for embedded devices [30].

However, since PUFs are physical structures that exploit side-effects in chip manufacturing, they typically possess a limited resilience against operational and environmental influences such as temperature, power supply variations or silicon aging effects. Hence, it is necessary to integrate effective error correction mechanisms at the prover while at the same time keeping the hardware overhead of their implementation minimal. Furthermore, existing security solutions based on PUFs typically require an extensive enrollment phase and maintaining large challenge/response verification databases.

Contributions. In this paper, we present the design and implementation of the ALU PUF, a novel minimalist hardware trust anchor based on manufacturing variations in commodity processors. We demonstrate the efficiency and effectiveness of the ALU PUF as a hardware trust anchor in PUFatt, a PUF-based (remote) attestation protocol. Our

DAC '14, June 01-05 2014, San Francisco, CA, USA

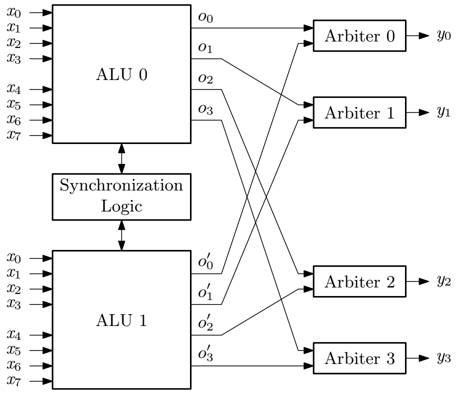

Figure 1: ALU PUF (example for 4-bit responses)

specific contributions are as follows.

ALU PUF Design. We present the ALU PUF, a novel PUF design based on the delay difference in two different ALUs[1] caused by manufacturing variations. The construction of ALU PUFs can be readily automated during the processor design phase by adding only a minimal number of components, i.e., a simple synchronization logic and some arbiters which are implementable by flip-flops.

ALU PUF based Remote Attestation Scheme. We propose PUFatt, a new lightweight remote attestation scheme built on top of the ALU PUF. Our attestation scheme prevents impersonation attacks through binding software-based attestation to hardware characteristics. Moreover, our trust anchor, the ALU PUF, is tightly-coupled with the processor architecture and thwarts hardware-based attacks.

Implementation and Evaluation. We implemented the ALU PUF in FPGA and show its low overhead, applicability, unclonability and stability.

Security Analysis. We analyze our PUF-based attestation scheme under realistic assumptions. We show that our attestation method is sound and secure against impersonation and basic hardware attacks.

2. THE ALU PUF

The ALU PUF exploits delay differences in identically designed and redundantly available logic components, such as the ALUs in a processor. Similar to the Arbiter PUF [7], the ALU PUF exploits the time difference a signal takes to travel along two symmetric delay paths within these components. These paths are identical by design and shall be the same by construction (layout), yet they incur different delays in practice due to intrinsic manufacturing process variations. Note that all the steps for the ALU PUF design can be readily automated and integrated within the design phase of the microprocessor. Automatable design-time optimizations are needed to ensure symmetry of the delay paths.

ALU PUF Design. For simplicity, we illustrate the ALU PUF design on an example with two 4-bit ALUs (Figure 1). Note that all modern processors contain redundancies in their ALU structure, resulting in low hardware overhead for implementation.

ALUs consist of integrated circuits for computing arithmetic and logic functions in hardware. In our example, each ALU takes multiple input signals $x = (x_0, \dots, x_7)$ (representing the ALU PUF challenge), guides them through a network of gates and wires (representing delay paths) and generates output signals $o = (o_1, \dots, o_3)$ and $o' = (o'_1, \dots, o'_3)$, respectively. Similar to the Arbiter PUF, the ALU PUF response $y = (y_0, \dots, y_3)$ is generated by arbiters depending on which ALU's output signals arrive first. The PUF responses are latched to special registers (implemetable by flip-flops) in the processor. To ensure that both ALUs are stimulated with the same input signals at exactly the same time, a simple synchronization logic is used. Hence, the ALUs are utilized both as ALUs *and* as a PUF by only requiring a minimal hardware overhead (i.e., flip-flops and synchronization logic). More specifically, our ALU PUF design is based on the carry propagation in ripple-carry adders, which are basic ALU components. Observe that the delay characteristics of the path from the inputs x_i and x_{i+4} to the outputs o_i and o'_i, respectively, depend on the inputs x_{i-1} and x_{i+3} for $1 \leq i \leq 3$ because carry bits in the ripple-carry adders are propagated from the LSB side to the MSB side of the full adders.

The ALU PUF is queried by using the `add` assembler instruction. More specifically, when the ALU is in the PUF mode (i.e., not in the general program execution mode), the `add` instruction reads the PUF challenge (operands) from the registers inside the CPU and performs the `add` operation. The time-difference between the availability of the resulting output bits computed by each ALU is then used to derive the PUF response. Depending on the operand bit-length of the adders in the ALU, we can easily build ALU PUFs with an arbitrary number of response bits.

In generic pipelined processor architectures, the memory access stage is the critical path [25]. Thus, integrating the components needed to use the ALUs as PUFs into the processor has only a negligible timing performance impact.

Security Objectives. The most important properties of PUFs and hence the (security) goals of the ALU PUF are *robustness*, *unclonability* and *unpredictability* [21, 1, 11]. Informally, robustness means that, when queried with the same challenge x multiple times, the PUF returns a similar response y with high probability. Physical unclonability demands that it is infeasible to produce two PUFs that cannot be distinguished based on their challenge/response behavior. Unpredictability requires that it is infeasible to predict the PUF response y to an unknown challenge x, even if the PUF can be adaptively queried a certain number of times.

PUF Response Verification. There are two approaches to verify the responses y of the ALU PUF: (1) using a database of challenge/response pairs (CRPs) recorded before deployment of the ALU PUF and (2) using an emulation $\mathsf{PUF.Emulate}()$ of the ALU PUF based on a simple PUF model H [22] (e.g., gate-level delay table lookups and delay additions) generated during the manufacturing process of the ALU PUF. The drawback of the database approach is its limited scalability since it requires storing a large number of CRPs for each PUF implementation. Further, due to the limited size of the database, this approach allows only for a limited number of authentications since CRPs should not be re-used to prevent replay attacks. The emulation-based approach overcomes these drawbacks but requires a protected interface to read out the gate-level delays required to emulate the PUF. This interface should be only accessible by a trusted entity (e.g., the PUF manufacturer) since otherwise the adversary could read out the gate delays and efficiently emulate the PUF, which would violate the unpredictability property. One approach to realize this interface in an ASIC implementation of the ALU PUF is to provide a test interface that can be permanently disabled by, e.g., using fuses. For our FPGA-based prototype emulation we did not implement such an interface as the gate-level delays were known.

Error Correction. Since we use the PUF responses as input to the attestation algorithm, we must correct errors

[1] Arithmetic and Logic Units (ALUs) are basic components in any processor.

in the PUF responses to maintain the verifiability of the attestation result. For this purpose, we adopt the low-cost error correction mechanism in [8]. In more detail, the PUF-enabled device $\mathcal{P}$ generates *helper data* h, which corresponds to the syndrome of the codeword of an error correcting code and enables the verifying party $\mathcal{V}$ to reconstruct the (noisy) response y used by $\mathcal{P}$. The only logic required at $\mathcal{P}$ is the syndrome generator of a linear block code, which performs a simple matrix multiplication. Our implementation of the syndrome generator uses the parity-check matrix of a BCH[32,6,16] code, which can correct up to 16 bit errors in a 32 bit PUF response using a $32 - 6 = 26$-bit helper data.

Response Obfuscation. It has been shown [27] that machine learning can be used to violate the unpredictability goal of most delay-based PUFs. Specifically, the adversary $\mathcal{A}$ can model the PUF based on a set of known CRPs that allows $\mathcal{A}$ to determine the response y to any challenge x for which $\mathcal{A}$ has never seen y before. To thwart these attacks, we employ an XOR-based obfuscation network (properly designed to prevent side-channels [28]).

The obfuscation works in two phases: In the first phase, the obfuscation is performed within the $2n$-bit PUF response y_0. Specifically, the i-th bit $y_0[i]$ and the $(i + n)$-th bit $y_0[i + n]$ of y_0 are XORed, resulting in an n-bit word a_0. That is $a_0[i] := y_0[i] \oplus y_0[i + n]$ for $1 \leq i \leq n$. The same is performed for a second PUF response y_1 to obtain an n-bit word a_1. These two words are then concatenated to a $2n$-bit word $b_0 := a_0||a_1$. The second obfuscation phase uses four $2n$-bit words $b_0, \ldots, b_3$ from the first obfuscation phase and XORs them, resulting in a $2n$-bit output $z := \bigoplus_{j=0}^{3} b_j$.

Note that the internal registers of the obfuscation network are not visible to the outside, i.e., the content of these registers cannot be accessed by code running on the processor. This ensures that the adversary cannot directly read the PUF responses and circumvent the obfuscation mechanism through any program running on the processor. Obfuscation must be performed after error correction to maintain verifiability of the outputs of the obfuscation network. This is because only a few bit errors in the input to the obfuscation network may incur a large number of output errors.

Architectural Support. To use the ALU both as ALU *and* PUF, we extend the instruction set of the microprocessor by two CPU instructions `pstart` and `pend` that start and stop the PUF operation of the ALU, respectively. `pstart` switches the ALUs into PUF mode. A simple synchronization logic between both ALUs ensures that both ALUs are triggered at the same time. After the PUF response has been generated, the `pend` instruction forwards the PUF response y to the post-processing logic and switches the processor back to normal mode. Since the PUF operation is performed in PUF mode, there is no performance impact on programs executed in normal mode.

3. ALU PUF-BASED ATTESTATION

PUFatt, our remote attestation scheme combines software-based attestation with PUFs by adapting the approach in [30]. The idea is to bind the attestation scheme to the prover hardware by entangling the attestation checksum computation with the PUF. This is done in a way that ensures outsourcing this computation to another device is infeasible due to the limited capacity of the communication interfaces of the prover. In the following, for brevity we write $\mathsf{PUF}()$ to denote the ALU PUF including the error correction mechanism and the obfuscation network.

System Model. In our attestation scheme a *prover* $\mathcal{P}$ reports the integrity of its (program) memory content S to a *verifier* $\mathcal{V}$. While $\mathcal{P}$ is an embedded device with constrained resources (e.g., a sensor node), $\mathcal{V}$ is a more powerful computing device (e.g., a smartphone or a laptop).

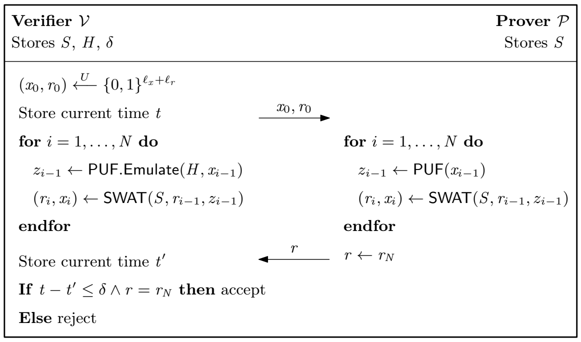
Verifier $\mathcal{V}$
Stores S, H, δ
Prover $\mathcal{P}$
Stores S
$(x_0, r_0) \xleftarrow{U} \{0,1\}^{\ell_x + \ell_r}$
Store current time t
x_0, r_0
for $i = 1, \ldots, N$ **do**
$z_{i-1} \leftarrow \mathsf{PUF.Emulate}(H, x_{i-1})$
$(r_i, x_i) \leftarrow \mathsf{SWAT}(S, r_{i-1}, z_{i-1})$
endfor
for $i = 1, \ldots, N$ **do**
$z_{i-1} \leftarrow \mathsf{PUF}(x_{i-1})$
$(r_i, x_i) \leftarrow \mathsf{SWAT}(S, r_{i-1}, z_{i-1})$
endfor
r
$r \leftarrow r_N$
Store current time t'
If $t - t' \leq \delta \wedge r = r_N$ **then** accept
Else reject

Figure 2: PUF-based attestation scheme

Security Objectives. An attestation scheme should achieve *correctness* and *soundness* [2]. Correctness informally means that an honest $\mathcal{P}$ whose memory content matches the memory content S expected by an honest $\mathcal{V}$ should always be accepted. Soundness informally means that any $\mathcal{P}$ with a program memory content different from S should succeed in making $\mathcal{V}$ accept only with negligible probability. Furthermore, the PUF-based attestation scheme should provide *prover authentication*, i.e., assure to $\mathcal{V}$ that the attestation algorithm has been computed by a particular $\mathcal{P}$.

Assumptions. As is common in the literature on software-based attestation [32, 31, 6, 10, 16, 15], we assume the attestation algorithm and its implementation used by $\mathcal{P}$ to be optimal in the sense that it is hard for the adversary to find another algorithm or implementation that can be executed by $\mathcal{P}$ in less time. Moreover, as in [30], we assume that the bandwidth of the communication interfaces of $\mathcal{P}$ is far lower than the bandwidth of the interface between the CPU and the PUF of $\mathcal{P}$. Further, we assume that $\mathcal{V}$ can emulate the PUF of $\mathcal{P}$, e.g., by knowing the gate-level delay table of the ALU PUF (Section 2).

Trust and Adversary Model. The adversary $\mathcal{A}$ can eavesdrop on and modify any data transmitted between $\mathcal{P}$ and $\mathcal{V}$. Further, $\mathcal{A}$ knows all data stored in the memory of $\mathcal{P}$ and can modify it. Note that by modifying the memory content of $\mathcal{P}$, $\mathcal{A}$ can change the program code and thus has full control of $\mathcal{P}$. However, as we show later in Section 4.1, $\mathcal{A}$ cannot clone $\mathsf{PUF}()$ of $\mathcal{P}$ and it is infeasible for $\mathcal{A}$ to predict the output z of $\mathsf{PUF}()$ to a new challenge x for which $\mathcal{A}$ has never seen z before. Moreover, any attempt of $\mathcal{A}$ to modify the hardware of $\mathcal{P}$ to enhance its computing and/or memory capabilities changes the challenge/response behavior of the PUF. Finally, $\mathcal{V}$ is trusted.

Protocol Specification. The PUFatt attestation scheme works as follows (Figure 2): The verifier $\mathcal{V}$ sends a random PUF challenge x_0 and a random *attestation challenge* r_0 to the prover $\mathcal{P}$. Based on these challenges and its memory content S, $\mathcal{P}$ iteratively computes the *attestation response* r using $\mathsf{PUF}()$ and the software attestation algorithm $\mathsf{SWAT}()$. Eventually, $\mathcal{P}$ sends r to $\mathcal{V}$, who accepts only if $\mathcal{P}$ responded within the expected time bound δ and if r matches the expected attestation response. Hereby, $\mathcal{V}$ recomputes r using an emulation $\mathsf{PUF.Emulate}()$ of $\mathcal{P}$'s $\mathsf{PUF}()$ (Section 2).

Instantiation. The attestation algorithm $\mathsf{SWAT}()$ can be instantiated based on any known software-based attestation scheme (e.g., [31, 37, 3]) with only minor modifications to integrate the ALU PUF. For our implementation, we adapted the algorithm in [31], which iteratively computes the attestation response r. As with most existing software-based attestation schemes, this algorithm samples a memory word of the memory S of the prover $\mathcal{P}$ in each round and uses it as

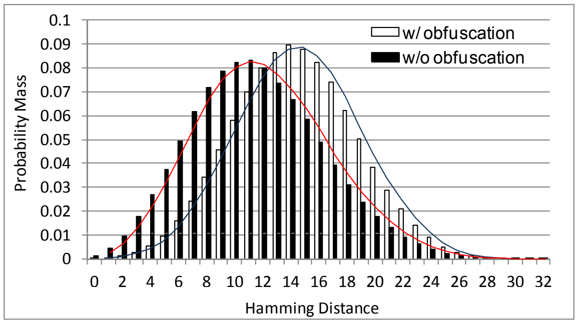

Figure 3: Inter-chip HD results

input to a compression function that iteratively computes r [2]. We adapted this algorithm to generate PUF challenges x_i and to take the output z_i of $\mathsf{PUF}()$ as additional input to the compression function in each round. Note that we omit the details on generating r due to limited space. For more details, please refer to the checksum algorithm in [31].

4. EVALUATION

4.1 Evaluation of the ALU PUF

Our evaluation results are based on a gate-level delay simulation of the ALU PUF. We leverage the delay model from [23] to calculate the gate-level delay under process variations. Our emulation uses the quad-tree process variation model [4], which assigns different threshold voltage variations to all gates in all simulated chips and targets the 45 nm technology node. Following [25], we assume that the distribution of the threshold voltage V_{th} in the chips follows a Gaussian distribution $N(\mu, \sigma^2)$ with $\frac{\sigma}{\mu} = 0.1$. We also present inter- and intra-chip distance measurement results of two ALU PUF implementations in two different FPGAs.

Unpredictability. We empirically assess the unpredictability of the ALU PUF by means of the Hamming distance (HD) between the responses y of *different* PUFs to the *same* challenge x (inter-distance) [21]. Specifically, we count the occurrences of each HD for $1,000,000$ different challenges x.

Our inter-chip HD results are presented in Figure 3, which shows both the inter-chip HD of the raw PUF responses (before obfuscation) and the inter-chip HD of the obfuscated PUF responses. The ideal inter-chip HD would be 16 bits (50%). Before and after obfuscation, the average inter-chip HD is 11.48 bits (35.9%) and 14.28 bits (44.6%), respectively. The ALU PUF shows a fairly good unpredictability which is comparable to other existing PUF designs [21], e.g., the Feed-forward Arbiter PUF (38% inter-chip HD) [17]. Further, as expected, the XOR-based obfuscation mechanism improves the unpredictability of PUF responses.

Robustness. We measure the robustness of the ALU PUF by means of the HD between the responses y of the *same* ALU PUF with regards to *same* challenge x under different operating conditions [21]. We consider three test cases that can affect the intra-chip HD: voltage variations, temperature variations and arbiter metastability. We examined voltage variations from 90% to 110% of the nominal ALU PUF supply voltage. Further, we consider operating temperatures between -20°C and $+120$°C. Again, HDs are are computed based on $1,000,000$ different challenges x.

The intra-chip HD results of the raw PUF responses (i.e., without error correction and obfuscation) are shown in Figure 4. The ideal intra-chip HD would be 0 bits (0%). The intra-chip HD between repetitive evaluations of the same ALU PUF is 3.62 bits (11.3%), which is comparable to other delay-based PUFs, e.g., the Feed-Forward Arbiter PUF (9.8%) [17]. The ALU PUF is quite robust even under extreme environmental conditions. This is because the ALUs' symmetric delay paths are very similarly affected, which compensates for the effect of the operating conditions. Since the redundant ALUs are in close proximity, the variations due to systematic spatial variations are also minimal. Hence, the main factor affecting the intra-chip HD is arbiter metastability. According to our simulation results and considering the error correction mechanism used, our PUF exhibits only a false negative rate of 1.53×10^{-07}, which is sufficient for most practical applications.

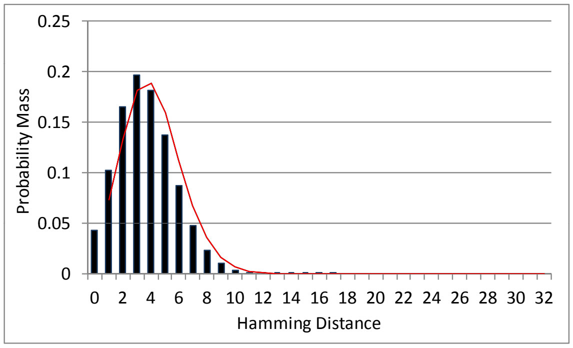

Figure 4: Intra-chip HD results under voltage and temperature variations and arbiter metastability

Table 1: FPGA implementation (16-bit ALU PUF)

Component	LUTs	Registers	XORs	BRAM	FIFO
ALU PUF	94	80	32	0	0
Synchronization logic	9	7	0	0	0
Syndrome generator	1,976	880	0	3	0
Obfuscation logic	224	0	0	0	0
PDL logic	4,096	128	0	0	0
SIRC logic	2,808	1,826	0	38	2

Implementation. We implemented the core parts of the ALU PUF in Xilinx Virtex 5 XC5VLX110T FPGA devices. The implemented PUF is based on a 16-bit model (instead of 32-bit) due to the resource constraints of the given FPGA. As pointed out in [20], implementing two completely symmetric delay paths in FPGA is challenging. Hence, we employed programmable delay lines (PDLs) to tune the PUF delay. Specifically, o_i and o'_i (where $1 \leq i \leq 16$) from ALU 0 and ALU 1 are passed through 64 stages of PDL switches to compensate for the skews in delays that occur due to the automated routing optimization performed by the Xilinx ISE tools. The blocks' placement is done manually to achieve the maximally achievable symmetry of the delay lines. The delay tuning process is carried out as described in [20]. By changing the PDL inputs, we calibrate the delay of the two symmetric delay paths so that on average the occurrence of 0 and 1 at each arbiter is about the same.

Table 1 summarizes the results from implementing our ALU PUF from scratch (i.e., when one does not re-use an existing ALU). The ALU PUF itself yields a significantly low hardware overhead compared to the supporting FPGA logic such as the PDL and SIRC logic which is used for PUF data collection [5]. However, when implementing the ALU PUF in ASIC, this supporting logic is not needed.

We also measured the inter- and intra-chip HDs of two 16-bit ALU PUF implementations on two different FPGA boards. The inter-chip HD is 3.0 (18.8%) and 6.6 (41.3%) bits without and with the XOR obfuscation, respectively. The intra-chip HD is 2.9 bits (18.6%), which is a little higher than in our simulation due to environmental fluctuations. The FPGA measurement results are consistent with our simulation results.

Side-channel Attack Resiliency. It has been shown [27] that most existing delay-based PUFs can be efficiently emulated using machine learning techniques, which violates the unclonability and unpredictability goals. We employ an XOR-based obfuscation network [34], which significantly increases the complexity of these attacks making them ineffective in practice [27]. It has been recently shown [18] that this approach to obfuscation can be attacked by combining side-channel analysis with machine learning techniques. However, several countermeasures are possible to prevent this attack [18, 28]. For instance, it is possible to minimize the side-channel leakage by making the timing behaviour and power consumption of the device independent of its inputs. Though we do not employ these techniques in our prototype PUF implementation, they can be easily deployed in our PUF architecture with a small hardware overhead.

4.2 Security of the Attestation Scheme

In this section, we evaluate the security of the PUF-based attestation scheme presented in Section 3. A formal analysis based on the security framework in [2] is planned as future work. Due to the robustness of the PUF (Section 4.1) and the error correction mechanism used (Section 2), $\mathsf{PUF}()$ (which includes the error correction mechanism and obfuscation network) always returns the same output z to the same challenge x. Further, we assume that the underlying software-based attestation scheme is sound and correct.

Soundness. Due to the correctness of the underlying software attestation scheme and the robustness of $\mathsf{PUF}()$, an honest prover $\mathcal{P}$ will always compute the correct attestation response r and be accepted by the verifier $\mathcal{V}$.

Correctness. Our PUF-based attestation scheme is based on a secure software attestation scheme whose compression function has been modified to take the output z of $\mathsf{PUF}()$ as an additional input using a similar approach as in [30]. This minor modification preserves the structure of the attestation algorithm and does not affect the correctness property of the underlying software attestation scheme [30].

Prover Authentication. The major objective of combining software-based attestation with PUFs is to assure to the verifier $\mathcal{V}$ that the attestation response r has been computed by the prover $\mathcal{P}$. There are two approaches for adversary $\mathcal{A}$ to violate prover authentication: (1) emulating $\mathsf{PUF}()$ of $\mathcal{P}$ on another device and (2) using $\mathsf{PUF}()$ of $\mathcal{P}$ as an oracle and computing r on another device. By assumption, $\mathcal{A}$ does not know the gate-delay table H of $\mathsf{PUF}()$ and, as discussed in Section 2, the most effective known emulation attacks [27] are infeasible in practice. Hence, it is infeasible for $\mathcal{A}$ to predict the correct outputs z_i of $\mathsf{PUF}()$ which are required to compute r. However, $\mathcal{A}$ could try to use $\mathsf{PUF}()$ of $\mathcal{P}$ as an oracle, meaning that whenever $\mathcal{A}$ needs an output z_i of $\mathsf{PUF}()$ for computing r, $\mathcal{A}$ may query $\mathsf{PUF}()$. Since by assumption, the bandwidth of the external communication interfaces of $\mathcal{P}$ is much lower than the bandwidth needed to transfer all z_i to $\mathcal{A}$ in time, $\mathcal{A}$ does not know at least one z_i. This means that $\mathcal{A}$ can predict the correct attestation response r only with the same probability as predicting the response of $\mathsf{PUF}()$, which according to the evaluation of the ALU PUF (Section 4.1) is very low.

Overclocking Attack Resiliency. One problem of purely software-based attestation schemes is that the adversary $\mathcal{A}$ could simply overclock the CPU of prover $\mathcal{P}$ to circumvent the enforcement of time-bound δ. Our ALU PUF-based attestation scheme is secure against this attack since the ALU PUF operates along with the CPU clock network. Thus, when $\mathcal{A}$ increases the clock frequency of the CPU, the clock cycle time will be reduced. This impacts the setup time T_{set} of the registers (flip-flops) used to store the raw PUF response y (i.e., before error correction and obfuscation). Specifically, the clock edge may reach the output flip-flops of the ALU before y is latched to them, resulting in wrong PUF responses. For correct PUF operation, the required condition is: $T_{\text{ALU}} + T_{\text{set}} < T_{\text{cycle}}$, where T_{cycle} is the duration of a clock cycle and T_{ALU} is the maximum signal propagation latency taken from the input flip-flop of the ALU to the output flip-flop of the ALU. When this condition is fulfilled, the ALU PUF will operate reliably. Thus, if $\mathcal{A}$ increases the clock frequency (hence, reducing the clock cycle time) such that T_{set} becomes too short, the ALU PUF will generate wrong responses. Hence, it is crucial to carefully set the clock frequency used for attestation.

Assume that $\mathcal{A}$ tries to hide the presence of malware at $\mathcal{P}$ by using a modified attestation algorithm. This modification will increase the number of clock cycles required to compute the correct attestation response r because $\mathcal{A}$ must perform additional computations to hide the presence of the malware. Now assume that $\mathcal{A}$ increases the clock frequency to stay within the time-bound δ of the attestation scheme, i.e., to compensate for the increased number of clock cycles taken by the modified attestation algorithm. This means that $\frac{C_{\mathcal{A}}}{C_{\mathsf{SWAT}}} < \frac{F_{\mathcal{A}}}{F_{\text{base}}}$, where $C_{\mathcal{A}}$ and C_{SWAT} are the number of clock cycles taken by the modified and the original attestation algorithm, respectively; $F_{\mathcal{A}}$ is the overclocked frequency used by $\mathcal{A}$ to circumvent the time-bound enforcement; and F_{base} is the default clock frequency of $\mathcal{P}$ expected by $\mathcal{V}$.

To thwart the overclocking attack, the base clock frequency $F_{\text{base}} = T_{\text{base}}^{-1}$ must be carefully chosen so that any attempt to increase the clock frequency of $\mathcal{P}$ results in a too short setup time T_{set}, and thus in wrong PUF responses. This implies that: $T_{\text{ALU}} + T_{\text{set}} < T_{\text{base}} < (T_{\text{ALU}} + T_{\text{set}}) \cdot \frac{F_{\mathcal{A}}}{F_{\text{ALU+set}}}$, where $F_{\text{ALU+set}}$ is the minimum required clock frequency for PUF operation and setup time. As long as T_{base} fulfills this condition, the ALU PUF will generate reliable responses. Otherwise, the attack will be detected by either wrong responses from the ALU PUF (due to the setup time violation) or the time-bound enforcement.

5. RELATED WORK

Timed Attestation. Timed attestation is a promising approach to verify the integrity of the software state of embedded systems. Previous work proposed to bind software-based attestation [32, 31, 6, 10, 16, 15] to the prover hardware by exploiting hardware-specific effects, such as caching behavior [12]. However, this was shown to be insufficient [33]. Other works used secure co-processors for more accurate time measurements [29] or validation of intermediate checksum results [10]. However, these approaches are not suitable for cost-effective embedded devices and cannot prevent proxy attacks. VIPER [16] uses a repeated, precisely timed sequence of attestation challenges in a local setting to assure that the checksum computation is not delegated to another device. PUFs have been proposed as trust anchors for attestation [30]. However, substantial practical issues such as the access speed of the PUF and the viability of querying the PUF in parallel to the software attestation algorithm remained unanswered in the earlier work.

Physically Unclonable Functions (PUFs). These primitives are an emerging technology for secure authentication and key storage. An overview of different PUF types, security definitions and requirements can be found in [21, 14, 1]. A processor architecture which supports secure execution and secure key storage based on PUFs was proposed in [35]. Processor-based PUFs that are triggered by CPU instructions were also proposed in [19]. However, this approach is vulnerable to environmental fluctuations since it is based on measuring the absolute latency differences across the chips instead of using relative latency differences, as in our design. In [13], a processor-based PUF structure was proposed, but the focus was on the aging-based response tuning and not software attestation.

Other PUFs using existing on-chip structures, e.g., memory-based PUFs [38, 9], have several limitations: (1) they only support a small number of challenge-response pairs which are used only for key generation and not for authentication, and (2) they can be queried only directly after the power-up, making it impractical to query them afterwards. In contrast our ALU PUF design leverages redundant components in microprocessors, which significantly reduces the hardware overhead required for implementation.

6. CONCLUSION

We present ALU PUF, a novel PUF design which enables a paradigm shift in secure software attestation. Our new PUF design is based on the delay differences in redundantly available components of microprocessors. ALU PUF is readily integrated as trust anchor into the PUF-based attestation scheme for embedded systems. We implemented the ALU PUF in FPGA and demonstrated its good statistical properties and low hardware overhead. The ALU PUF-based attestation scheme enables the secure (remote) attestation of embedded devices and, in contrast to previous approaches, allows impersonation attacks to be detected. Further, due to tight coupling of the ALU PUF and the processor architecture, our PUF prevents overclocking attacks, which are a threat to purely software-based attestation schemes. Our new scheme provides a low-cost attestation mechanism tailored to the resource-constraints of lightweight embedded systems.

Acknowledgements. This work was supported in parts by NSF career grants (No. 0644289 and 1116858) and ARO YIP grant (R17450).

7. REFERENCES

[1] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, and C. Wachsmann. A formalization of the security features of physical functions. In *IEEE Symposium on Security and Privacy (S&P)*, 2011.

[2] F. Armknecht, A.-R. Sadeghi, S. Schulz, and C. Wachsmann. A security framework for the analysis and design of software attestation. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[3] Y.-G. Choi, J. Kang, and D. Nyang. Proactive code verification protocol in wireless sensor network. In *Computational Science and Its Applications (ICCSA)*, 2007.

[4] B. Cline, K. Chopra, D. Blaauw, and Y. Cao. Analysis and modeling of CD variation for statistical static timing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2006.

[5] K. Eguro. SIRC: An extensible reconfigurable computing communication API. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010.

[6] R. W. Gardner, S. Garera, and A. D. Rubin. Detecting code alteration by creating a temporary memory bottleneck. *IEEE Transactions on Information Forensics and Security*, 2009.

[7] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In *ACM Conference on Computer and Communications Security (CCS)*, 2002.

[8] A. Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann. Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs. In *Financial Cryptography and Data Security (FC)*. 2012.

[9] D. Holcomb, W. Burleson, and K. Fu. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers*, 58(9), 2009.

[10] M. Jakobsson and K.-A. Johansson. Retroactive detection of malware with applications to mobile platforms. In *Workshop on Hot Topics in Security (HotSec)*, 2010.

[11] S. Katzenbeisser, Ünal Kocabaş, V. Rozic, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann. PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2012.

[12] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *USENIX Security Symposium*, 2003.

[13] J. Kong and F. Koushanfar. Processor-based strong physical unclonable functions with aging-based response tuning. *IEEE Transactions on Emerging Topics in Computing*, PP(99), 2013.

[14] F. Koushanfar and A. Mirhoseini. A unified framework for multimodal submodular integrated circuits Trojan detection. *IEEE Transactions on Information Forensics and Security*, 2011.

[15] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth. New results for timing-based attestation. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[16] Y. Li, J. M. McCune, and A. Perrig. VIPER: Verifying the integrity of PERipherals' firmware. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[17] R. Maes and I. Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In *Towards Hardware-Intrinsic Security*. Springer, 2010.

[18] A. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar. Combined modeling and side channel attacks on strong PUFs. *ePrint*, 2013.

[19] A. Maiti and P. Schaumont. A novel microprocessor-intrinsic physical unclonable function. In *Field Programmable Logic and Applications (FPL)*, 2012.

[20] M. Majzoobi, F. Koushanfar, and S. Devadas. FPGA PUF using programmable delay lines. In *Information Forensics and Security (WIFS)*, 2010.

[21] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Techniques for design and implementation of secure reconfigurable PUFs. *ACM TRETS*, 2(1), 2009.

[22] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas. Slender PUF protocol: A lightweight, robust, and secure authentication by substring matching. In *IEEE Symposium on Security and Privacy Workshops (SPW)*, 2012.

[23] D. Markovic, C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey. Ultralow-power design in near-threshold region. In *Proceedings of the IEEE*, 2010.

[24] J. Nick L. Petroni, T. Fraser, J. Molina, and W. A. Arbaugh. Copilot — A coprocessor-based kernel runtime integrity monitor. In *USENIX Security Symposium*, 2004.

[25] Y. Pan, J. Kong, S. Ozdemir, G. Memik, and S. W. Chung. Selective wordline voltage boosting for caches to manage yield under process variations. In *Design Automation Conference (DAC)*, 2009.

[26] B. Parno, J. M. McCune, and A. Perrig. Bootstrapping trust in commodity computers. In *IEEE Symposium on Security and Privacy (S&P)*, 2010.

[27] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[28] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, F. Koushanfar, and W. Burleson. Power and timing side channels for pufs and their efficient exploitation. *IACR Cryptology ePrint Archive*, 2013.

[29] D. Schellekens, B. Wyseur, and B. Preneel. Remote attestation on legacy operating systems with Trusted Platform Modules. *Science of Computer Programming*, 2008.

[30] S. Schulz, A.-R. Sadeghi, and C. Wachsmann. Short paper: Lightweight remote attestation using physical functions. In *ACM Conference on Wireless Network Security (WiSec)*, 2011.

[31] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *ACM Workshop on Wireless security (WiSe)*, 2006.

[32] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

[33] U. Shankar, M. Chew, and J. D. Tygar. Side effects are not sufficient to authenticate software. In *USENIX Security Symposium*, 2004.

[34] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Design Automation Conference (DAC)*, 2007.

[35] G. E. Suh, C. W. O'Donnell, and S. Devadas. AEGIS: A single-chip secure processor. *Information Security Technical Report*, 2005.

[36] Trusted Computing Group (TCG). *TPM Spec.*, 2004.

[37] Y. Yang, X. Wang, S. Zhu, and G. Cao. Distributed software-based attestation for node compromise detection in sensor networks. In *Symposium on Reliable Distributed Systems (SRDS)*, 2007.

[38] Y. Zheng, M. Hashemian, and S. Bhunia. RESP: A robust physical unclonable function retrofitted into embedded SRAM array. In *Design Automation Conference (DAC)*, 2013.