Short Paper: Lightweight Remote Attestation using Physical Functions

Ahmad-Reza Sadeghi
TU Darmstadt (CASED)
& Fraunhofer SIT
Darmstadt, Germany
ahmad.sadeghi@trust.cased.de

Steffen Schulz
TU Darmstadt (CASED)
& Macquarie University (INSS)
Darmstadt, Germany
steffen.schulz@cased.de

Christian Wachsmann
TU Darmstadt (CASED)
Darmstadt, Germany
christian.wachsmann@cased.de

ABSTRACT

Remote attestation is a mechanism to securely and verifiably obtain information about the state of a remote computing platform. However, resource-constrained embedded devices cannot afford the required trusted hardware components, while software attestation is generally vulnerable to network and collusion attacks.

In this paper, we present a lightweight remote attestation scheme that links software attestation to remotely identifiable hardware by means of Physically Unclonable Functions (PUFs). In contrast to existing software attestation schemes, our solution (1) resists collusion attacks, (2) allows the attestation of remote platforms, and (3) enables the detection of hardware attacks due to the tamper-evidence of PUFs.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Physical security, invasive software (e.g., viruses, worms, Trojan horses)

General Terms

Design, Security

Keywords

Remote Attestation, Software-based Attestation, Physically Unclonable Functions (PUFs), Embedded Devices

1. INTRODUCTION

One of the major challenges in computer security is how to gain assurance that a local or remote computing platform behaves as expected. Various approaches have been proposed that aim to assure the correct and secure operation of computer systems (*attestation*) [15]. Common to all existing approaches is that the platform to be evaluated (*prover*) sends a status report of its current configuration to a *verifier* to demonstrate that it is in a known and thus trustworthy state. Since malicious hard- or software on the prover's platform may forge this report, its authenticity is typically assured by a secure co-processor [5, 12] or trusted software [1].

A recent industrial initiative towards the standardization of attestation was brought up by the Trusted Computing Group (TCG) by specifying the Trusted Platform Module (TPM) [22] as a trust anchor for authentic reporting of a platform's software state. Today, TPMs are typically implemented as secure co-processors and are available in many PCs, laptops, and server systems. The TCG also specifies the Mobile Trusted Module (MTM) [23], which is a TPM for mobile and embedded devices. However, the integration of security hardware in low-cost embedded devices (e.g., wireless sensor nodes) is often infeasible. in this context, *software attestation* [20] was proposed, requiring neither trusted hardware nor a secure software core.

Software attestation exploits the computational limits of the prover to ensure that only a specific algorithm can be executed within a certain time frame. Within this algorithm, the prover computes a *checksum* of its software state, e.g., its program memory content, and sends it to the verifier. The verifier computes a reference checksum using a reference software state and accepts the prover only if (1) the checksum reported by the prover matches the reference checksum and (2) the prover answered within the same time an honest device would have needed. The first check guarantees that the expected software is present at the prover, while the second ensures that the prover has not performed additional computations, e.g., to hide malicious software.

Unfortunately, software attestation schemes require additional assumptions to be secure, namely (1) the prover cannot be impersonated by or collude with other, potentially more powerful, devices, and (2) the hardware of the prover was not modified to increase its performance. As a result, the existing software attestation schemes are unsuitable for remote attestation or in scenarios where the adversary can modify the prover's hardware, such as sensor networks.

To overcome these problems the checksum must be linked to the prover's platform. One possible solution links the checksum computation to hardware-specific side-effects, such as CPU states and caching effects that are considered to be expensive to simulate [9]. However, it has been shown that these side-effects are not appropriate to achieve a strong link to the underlying hardware [21, 11] as they only bind the software computation to *classes* of devices instead of individual provers.

[*] Full version available upon request.

WiSec'11, June 14–17, 2011, Hamburg, Germany.
Copyright 2011 ACM 978-1-4503-0692-8/11/06 ...$10.00.

Contribution.

In this paper, we propose a lightweight remote attestation scheme that combines software attestation with device-specific hardware functions. Specifically, we show how Physically Unclonable Functions (PUFs) can be integrated into the software attestation s.t. a compromised device is unable to efficiently outsource the software checksum computation to colluding parties and propose practical optimizations to facilitate the verification of the PUF.

In contrast to plain software attestation, our scheme (1) is secure against a collusion of malicious provers, (2) allows for the authentication and attestation of remote provers, and (3) enables the detection of hardware attacks on the prover. We present different solutions for the efficient and practical verification of PUFs by the verifier and discuss their trade-offs. The proposed scheme is applicable to any current (and likely future) software attestation protocol.

2. Physically Unclonable Functions (PUFs)

A Physically Unclonable Function (PUF) is a noisy function that is embedded into a physical object, e.g., an integrated circuit [14]. Today, there are already several PUF-based security products aimed for the market, e.g., PUF-enabled RFID chips and proposals for IP-protection and anti-counterfeiting solutions [26, 8]. When queried with a *challenge* x, a PUF generates a *response* $y \leftarrow \texttt{PUF}(x)$ that depends on both, x and the unique device-specific intrinsic physical properties of the object containing PUF. Since PUFs are subject to noise (e.g., thermal noise), they typically return slightly different responses when queried with the same challenge multiple times. However, these output variations can be eliminated by using *fuzzy extractors* [4], which can be efficiently implemented on resource-constrained devices [24]. Hence, PUFs can be used as deterministic functions.

Based on [2, 18], we consider PUFs that have the following properties, where PUF and PUF′ are two different PUFs:

- *Robustness:* When queried with the same challenge x, PUF always returns the same response y.
- *Independence:* When queried with the same challenge x, PUF and PUF′ return different responses y and y'.
- *Pseudo-randomness:* It is infeasible to distinguish a PUF from a pseudo-random function PRF.
- *Tamper-evidence:* Any attempt to physically access the object containing PUF irreversibly changes PUF, i.e., PUF cannot be evaluated any more but is turned into a random $\texttt{PUF}' \neq \texttt{PUF}$.

Independence and pseudo-randomness imply that $\mathcal{A}$ cannot predict PUF responses to unknown challenges, which means that $\mathcal{A}$ cannot simulate a PUF based on its challenge-response behavior. Moreover, tamper-evidence ensures that $\mathcal{A}$ cannot obtain any information on the PUF by physical means, e.g., hardware attacks. Hence, $\mathcal{A}$ cannot simulate or clone a PUF.

3. PUF-BASED ATTESTATION

Our PUF-based attestation scheme extends existing software attestation protocols. A software attestation protocol is a two-party protocol between a *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$, where $\mathcal{V}$ should be convinced that $\mathcal{P}$ is in a trusted software *state* S. Typically, $\mathcal{P}$ is an embedded device with constrained computing capabilities (e.g., a sensor node), whereas $\mathcal{V}$ is a more powerful computing device (e.g., a base station). On a high level, all known software attestation protocols exploit the computational limits of $\mathcal{P}$ to assure that nothing else than a specific trusted algorithm can be executed within a specific time frame.

In contrast to existing software attestation schemes, our solution assures the verifier $\mathcal{V}$ that the attestation result has actually been computed by the original hardware of a specific prover $\mathcal{P}$. We propose to use a hardware checksum based on PUFs to include device-specific properties of $\mathcal{P}$'s hardware into the attestation protocol. Our design exploits the limited throughput of external interfaces to prevent an adversary from outsourcing the computation of the software checksum to a more powerful computing device.

Trust model and assumptions.

The adversary $\mathcal{A}$ controls the communication between the verifier $\mathcal{V}$ and the prover $\mathcal{P}$, i.e., $\mathcal{A}$ can eavesdrop, manipulate, reroute, and delete all messages sent by $\mathcal{V}$ and $\mathcal{P}$. Moreover, $\mathcal{A}$ knows all algorithms executed by $\mathcal{P}$ and can install malicious software on $\mathcal{P}$. However, due to the unclonability of the PUF (Section 2), $\mathcal{A}$ cannot simulate the hardware checksum, while the tamper-evidence of the PUF ensures that $\mathcal{A}$ cannot physically access or manipulate the internal interfaces between CPU, memory, and PUF of $\mathcal{P}$. Further, we assume that external interfaces of $\mathcal{P}$ are significantly slower than the internal interface that is used by the CPU to access the hardware checksum. All provers $\mathcal{P}$ are initialized in a secure environment before deployment. The verifier $\mathcal{V}$ is trusted to behave according to the protocol. Moreover, $\mathcal{V}$ can simulate any algorithm that can be executed by $\mathcal{P}$ in real time and maintains a database D containing the identity I and the exact hard- and software configuration of each $\mathcal{P}$.

Protocol description.

Figure 1 shows the proposed PUF-based attestation protocol, consisting of a generalized software-attestation protocol with additional inclusion of a device-characteristic *hardware checksum*[1] function HwSum() at the prover $\mathcal{P}$ and EmulateHwSum() at the verifier $\mathcal{V}$. By careful integration of this hardware checksum into the software attestation algorithm, we bind the software attestation to the respective hardware platform, enabling true remote attestation.

The main protocol is the generalization of a typical software attestation protocol: The verifier $\mathcal{V}$ starts the protocol by sending a random challenge r to the prover $\mathcal{P}$ and then measures the time $\mathcal{P}$ takes to reply with the checksum σ_k computed over its current software state S (e.g., its program memory). In detail, on receipt of r, $\mathcal{P}$ sets up the initial checksum value σ_0 and Pseudo-Random Number Generator (PRNG) state r as required by the under-

[1] For the purpose of this paper, we consider HwSum() to be a PUF to gain tamper evidence, however, simpler implementations are possible, e.g., an HMAC with a hard-wired key.

Verifier $\mathcal{V}$		**Prover** $\mathcal{P}$
Stores $D = \{\ldots, (I, S_I, C_I, \delta_I), \ldots\}$		Stores (I, S)
Choose random challenge r		
Save current time t	$\xrightarrow{r}$	
$r'_0 \leftarrow r$		$r_0 \leftarrow r$
$\sigma'_0 \leftarrow \texttt{InitSwSum}(r_0)$		$\sigma_0 \leftarrow \texttt{InitSwSum}(r_0)$
for $i = 1$ to k **do**		**for** $i = 1$ to k **do**
$y'_i \leftarrow \texttt{EmulateHwSum}(C_I, \sigma'_{i-1})$		$y_i \leftarrow \texttt{HwSum}(\sigma_{i-1})$
$(a'_i, r'_i) \leftarrow \texttt{GenMemAddr}(r'_{i-1}, y'_i)$		$(a_i, r_i) \leftarrow \texttt{GenMemAddr}(r_{i-1}, y_i)$
$\sigma'_i \leftarrow \texttt{SwSum}(\sigma'_{i-1}, S_I[a'_i])$		$\sigma_i \leftarrow \texttt{SwSum}(\sigma_{i-1}, S[a_i])$
end		**end**
Save current time t'	$\xleftarrow{\sigma_k}$	
if $(t' - t) \leq \delta_j$ **and** $\sigma_k = \sigma'_k$ **then accept** $\mathcal{P}$		
else reject $\mathcal{P}$		

Figure 1: Remote attestation based on physical functions

lying software attestation scheme. $\mathcal{P}$ then iteratively computes σ_k by taking i random measurement samples out of S. Specifically, in each iteration i of the checksum computation $\mathcal{P}$ invokes three procedures: `GenMemAddr`(), `SwSum`(), and `HwSum`(). `GenMemAddr`(r_{i-1}, y_i) is used to generate an output r_i and a memory address a_i, which determines the next memory block $S[a_i]$ of S to be included into the software checksum as $\sigma_i \leftarrow \texttt{SwSum}(\sigma_{i-1}, S[a_i])$. Note that `SwSum`() is the same function as in plain software attestation, while we require only a minor modification of `GenMemAddr`() to include the hardware checksum output y_i. Typically, modern software attestation schemes implement `GenMemAddr`() as a Pseudo-Random Number Generator (PRNG) to prevent efficient pre-computation or memory mappings attacks. However, neither the PRNG nor the `SwSum`() are required to be cryptographically strong [20]. Hence, it is usually straightforward to integrate y_i into `GenMemAddr`() by using it as an additional seed to the PRNG.

In contrast to plain software attestation, our attestation scheme integrates a hardware checksum `HwSum`() into each iteration i, yielding the previously mentioned additional input $y_i \leftarrow \texttt{HwSum}(\sigma_i)$ to the `GenMemAddr`() procedure. As a result, every iteration of the software checksum additionally depends on the result of the device-characteristic hardware checksum, thus binding the attestation response σ_k to the prover's hardware. Similarly, each iteration of `HwSum`() depends on the previous intermediate software checksum σ_{i-1}, s.t. `HwSum`() cannot be executed independently of `SwSum`(). However, we emphasize that the depicted algorithm can be optimized to execute `HwSum`() and `SwSum`() in parallel in all but the very first iteration.

After every memory block $S[a_i]$ has been included into the checksum at least once, $\mathcal{P}$ sends σ_k to $\mathcal{V}$. While waiting for the response of $\mathcal{P}$, $\mathcal{V}$ can compute a reference checksum σ'_k by simulating the computation of $\mathcal{P}$ using the known trusted software state S_I recorded in database D and emulate `HwSum`() using `EmulateHwSum`() with some verification data C_I, which is secret information only available to $\mathcal{V}$. $\mathcal{V}$ accepts only if (1) $\mathcal{P}$ replied within a certain time frame δ_I and (2) σ_k matches σ'_k. The first check ensures that $\mathcal{P}$ computed σ_k in about the same time δ_I an honest device would have needed and has not performed additional computations, e.g., to hide the presence of malware. The second check verifies whether the software state S measured by $\mathcal{P}$ corresponds to the known trusted software state S_I. If either of these checks fails, $\mathcal{P}$ is assumed to be in an unknown software state and is rejected.

Note that the verification of the PUF-based hardware checksum by $\mathcal{V}$ is not straightforward: $\mathcal{V}$ must be able to predict the outputs of the PUF, while this must be infeasible for $\mathcal{A}$. This is further complicated by the large amount of hardware checksum responses required by our construction and the closely parallelized execution of software and hardware checksum. Hence, the integration of PUFs into software attestation requires careful consideration and we discuss possible instantiations in Section 4

Security objectives.

In contrast to existing software attestation schemes, our PUF-based attestation scheme additionally achieves the following security goals:

- *Correctness:* A prover in a known trusted state must always be accepted by the verifier.
- *Unforgeability:* A prover in an unknown state must be rejected by the verifier. Note that this also includes attacks, where the adversary makes the sensor node to collude with more powerful devices to forge the attestation.
- *Prover authentication:* A prover pretending to be another prover must be rejected by the verifier.
- *Prover isolation:* A prover colluding with other (malicious) provers must be rejected by the verifier.
- *Tamper-evidence:* A prover that is not in its original hardware state must be rejected by the verifier.

4. INSTANTIATION

In this section, we show how existing software attestation schemes can be used to instantiate software checksum `SwSum`() and the memory address generator `GenMemAddr`() with only minor modifications. Moreover, we discuss different instantiations of the hardware checksum `HwSum`() and, in particular, the corresponding secret verification data C_I and `EmulateHwSum`() algorithm.

4.1 Memory Address Generation and Software Checksum

The memory address generator $\texttt{GenMemAddr}()$ and the software checksum $\texttt{SwSum}()$ components of our PUF-based attestation scheme can be instantiated using any of the existing software-based attestation schemes (e.g., [19, 27, 3]) with only minor modifications to $\texttt{GenMemAddr}()$ for the integration of the hardware checksum $\texttt{HwSum}()$. In all modern software attestation designs, $\texttt{GenMemAddr}()$ is implemented as a PRNG with internal state r that is used to generate pseudorandom outputs a_i. We can thus integrate the output y_i of $\texttt{HwSum}()$ simply by merging it with the current state r in each iteration. Due to the unpredictability property of the PUF (Section 2), this is equivalent to (partly) reseeding the PRNG, which effectively prevents the PRNG from repeating its sequence.

4.2 Hardware Checksum

We present two alternative instantiations of the hardware checksum $\texttt{HwSum}()$ based on emulatable and non-emulatable PUFs. In general, emulatable PUFs yield more efficient protocols. However, since PUFs are not expected to be emulatable by design (Section 2), we focus on solutions for different approaches based on non-emulatable PUFs.

4.2.1 *Emulatable PUFs*

One approach to implement $\texttt{HwSum}()$ are emulatable PUFs, which allow the manufacturer of the PUF to set up a mathematical model that enables the prediction of PUF responses to unknown challenges [13, 16]. Typically, the creation of this model requires extended access to the PUF hardware, which is only available during the manufacturing process of the PUF and permanently disabled before deployment [13].

More detailed, during the production of the hardware of prover $\mathcal{P}$, the trusted hardware manufacturer sets up a secret mathematical model C_I of $\texttt{PUF}()$. Before deployment of $\mathcal{P}$, the interface for modelling the $\texttt{PUF}()$ is then disabled s.t. any attempt to reactivate it leads to an irreversible change of $\texttt{PUF}()$. During deployment of $\mathcal{P}$, C_I and an algorithm $\texttt{EmulateHwSum}()$ for emulating $\texttt{HwSum}()$ is given to the verifier $\mathcal{V}$. In the attestation protocol, $\mathcal{P}$ computes $\texttt{HwSum}(\cdot) = \texttt{PUF}(\cdot)$, whereas $\mathcal{V}$ emulates $\texttt{HwSum}(\cdot) = \texttt{EmulateHwSum}(C_I, \cdot)$.

In practice, emulatable PUFs can be realized by most delay-based PUFs (e.g., Arbiter PUFs [10, 7] and Ring Oscillator PUFs [6]), which allow for creating precise mathematical models based on machine learning techniques [17]. However, the security properties of practical instantiations of emulatable PUFs still need further evaluation. Hence, in the following section, we present different solutions based on non-emulatable PUFs.

4.2.2 *Non-emulatable PUFs*

For non-emulatable PUFs, the verifier $\mathcal{V}$ typically maintains a secret database D of PUF challenges and responses, called Challenge Response Pair (CRP) database. Note that our attestation scheme requires PUFs that ideally have an exponentially large CRP space, such that an adversary $\mathcal{A}$ with direct access to the PUF cannot create a complete CRP database and then emulate the PUF. However, this means that the verifier $\mathcal{V}$ can also store a subset of the CRP space. We thus have to deterministically limit the CRP subspace used during attestation without allowing the adversary to exploit this to simulate the PUF in future attestation runs.

In the following, we describe two different approaches of how non-emulatable PUFs can be used to instantiate $\texttt{HwSum}()$.

Commitment to procedure.

One approach is creating a database D of attestation challenge messages (q, r) and the corresponding checksums σ_k in a secure environment before the prover $\mathcal{P}$ is deployed. In the attestation protocol, the verifier $\mathcal{V}$ can then use D to obtain the reference checksum σ_k instead of emulating the PUF.

Specifically, before deployment, $\mathcal{V}$ runs the attestation protocol several times with $\mathcal{P}$. For each protocol run, $\mathcal{V}$ records in D the attestation challenge (r, q) sent to $\mathcal{P}$ and the corresponding checksum σ'_k returned by $\mathcal{P}$. When running the attestation protocol after deployment, $\mathcal{V}$ chooses a random set $(I, (r, q), \sigma'_k) \in D$ and sends (r, q) to $\mathcal{P}$, which then computes σ_k using $\texttt{HwSum}(\cdot)$. $\mathcal{V}$ accepts $\mathcal{P}$ only if $\mathcal{P}$ replied with $\sigma_k = \sigma'_k$ in time δ_I.

The solution allows for very efficient verification of σ_k by $\mathcal{V}$, however, the number of attestation protocol runs of each $\mathcal{P}$ is limited by the size of D. Moreover, this approach does not allow to update the software state of $\mathcal{P}$ after deployment, e.g., to fix bugs that allow runtime compromise.

Commitment to challenge.

Since updates to the software of the prover $\mathcal{P}$ are usually developed after deployment of $\mathcal{P}$, the software state S and thus the inputs to $\texttt{HwSum}()$ are not known before deployment of $\mathcal{P}$ and the final checksum value σ_k cannot be computed in advance.

Our solution to this problem is to reduce the amount of challenges x_i generated by the intermediate checksum results σ_i, s.t. it becomes feasible to create a CRP database independently of σ_i, and thus S. To prevent the adversary from exploiting this to simulate the attestation procedure, we use a random offset q to determine this reduced CRP space within the overall CRP space of $\texttt{HwSum}()$, such that the adversary cannot generate the required CRPs before the actual attestation protocol starts. The offset q is sent from the verifier $\mathcal{V}$ to $\mathcal{P}$ together with the random attestation challenge r in the first message of the attestation protocol (Figure 1).

More detailed, we chose $f(\cdot)$ to be a function that maps intermediate checksum results σ_i to bitstrings of length n and derive the challenges as $x_i \leftarrow \texttt{HwSum}(q||f(\sigma_i))$. Before deployment, the verifier $\mathcal{V}$ then evaluates $y_j \leftarrow \texttt{HwSum}(q||j)$ for $j \in \{0, \ldots, 2^n - 1\}$, and records $(q, y_0, \ldots, y_{2^n-1})$ in $C_{I,q}$ for a number of randomly chosen offsets q.

After deployment, $\mathcal{V}$ chooses a random nonce r and an offset $q \in C_I$ to start an attestation. The prover $\mathcal{P}$ then computes the checksum σ_k using $\texttt{HwSum}(q||\sigma_{i-1})$. While waiting for the response of $\mathcal{P}$, $\mathcal{V}$ computes the reference checksum σ'_k using $\texttt{EmulateHwSum}(C_{I,q})$ and the current reference software state S_I. $\mathcal{V}$ accepts only if $\mathcal{P}$ replied with $\sigma_k = \sigma'_k$ in time δ_I.

In this approach, the number of attestations are limited by the amount of random offsets q for which a CRP subspace has been generated in advance and by the storage available at the verifier $\mathcal{V}$. The offsets cannot be re-used since they cannot be encrypted and are potentially disclosed to the adversary.

On-demand CRP generation.

As a final modification, we propose a method to reduce the storage requirements at the verifier $\mathcal{V}$ and to allow a theoretically unlimited number of attestation protocols runs, by generating additional CRP subspaces on demand once an attestation succeeded.

Specifically, $\mathcal{V}$ and $\mathcal{P}$ can establish a mutually authenticated and confidential channel after successful attestation to exchange additional CRPs for future attestation runs. For this purpose, σ_k is treated as a common shared secret and the last message shown in the attestation protocol in Figure 1 is replaced with explicit key confirmation.

$\mathcal{V}$ can then send a new random offset q to $\mathcal{P}$, who responds with a response vector $y_j \leftarrow \mathtt{PUF}(q||j)$ for $j \in \{0, \ldots, 2^n - 1\}$ sorted by j. Finally, $\mathcal{P}$ deletes q and y_j from its memory and $\mathcal{V}$ updates $C_I \leftarrow (q, y_0, \ldots, y_{2^n-1})$ accordingly.

Note that this approach doubles the computational load of $\mathcal{P}$ and increases the communication load, so that it may not be suitable for, e.g., sensor networks.

5. SECURITY CONSIDERATIONS

In the following, we show that our PUF-based attestation protocol presented in Section 3 achieves prover authentication and prover-isolation. Hereby, we assume the underlying software attestation schemes and PUFs to fulfill their security properties.

Correctness and unforgeability of attestation.

Our solution preserves the security of existing software attestation schemes, consisting of the GenMemAddr() and SwSum() procedures. Our modifications are limited to GenMemAddr(), where we add the pseudo-random PUF responses y_i as an additional input to the PRNG state update procedure. Done properly, additional input to the PRNG state update will, in the worst case, not increase but keep the entropy of the internal PRNG state when compared with the regular PRNG state update. The required modifications thus do not affect correctness and unforgeability since the output distribution of the original and the modified GenMemAddr() procedure remain computationally indistinguishable as long as the original PRNG is secure.

Prover identification.

The main security goal of our design is to link the checksum to the hardware of the prover $\mathcal{P}$. Our solutions achieves this goal by identifying $\mathcal{P}$ based on the outputs of the hardware checksum HwSum(). The implementation of this requirement is straightforward: We must ensure that a sufficient amount of identifying information is generated by HwSum() and incorporated into the attestation checksum σ_k to prevent simple guessing attacks.

Prover isolation.

Our design runs the software and hardware checksums SwSum() and HwSum() in parallel and creates a strong algorithmic dependence on the output of both checksums in the respective previous iteration. To detach the computation of SwSum() from the hardware of the prover $\mathcal{P}$, the adversary $\mathcal{A}$ must thus simulate the function $y_{i-1} \leftarrow \mathtt{HwSum}()$ for each iteration i of the software checksum to generate the correct input to the memory generator GenMemAddr(). Furthermore, the intermediate checksum results σ_i are used as input to the next iteration of HwSum(). Hence, there are three major obstacles for $\mathcal{A}$: (1) the performance of HwSum() cannot be increased due to the tamper-evidence of the PUF, (2) $\mathcal{A}$ must involve the original hardware of $\mathcal{P}$ due to the unclonability of the PUF, and (3) the minimum *additional* delay incurred by transferring the HwSum() input and output bytes to a remote device is dictated by the throughput of the external communication interfaces of $\mathcal{P}$, since $\mathcal{A}$ cannot access the significantly faster internal interface between the CPU and HwSum(), which can be protected by the PUF.

Hence, any attempt to run HwSum() and SwSum() on separate devices will significantly increase the time required for all HwSum() iterations, regardless of the gained performance improvement on the SwSum() computation.

6. CONCLUSION

We presented a novel approach to attest both the software and the hardware configuration of a remote platform for embedded devices, which do not possess trusted hardware components. Our solution combines existing software attestation with cost-efficient physical security primitives, Physically Unclonable Functions (PUFs). In contrast to existing software attestation protocols, our scheme does not require an authenticated channel between the prover and the verifier and reliably prevents remote provers from colluding with other systems to forge the software checksum. We are currently working on an prototype implementation.

Acknowledgement

This work has been supported in part by the European Commission under grant agreement ICT-2007-238811 UNIQUE and ICT-2007-216676 ECRYPT NoE phase II.

7. REFERENCES

[1] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 65–71, Oakland, CA, May 1997. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[2] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009*, volume 5912, chapter 40, pages 685–702. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[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*, pages 1085–1096. Springer, August 2007.

[4] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology – EUROCRYPT '2004*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2004.

[5] J. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the IBM 4758 Secure Coprocessor. *IEEEC*, 34(10):57–66, 2001.

[6] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In *ACM Conference on Computer and Communications Security*, pages 148–160, New York, NY, USA, 2002. ACM Press.

[7] B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas. Identification and authentication of integrated circuits: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(11):1077–1098, 2004.

[8] Intrinsic ID. Intrinsic id — product page. `http://www.intrinsic-id.com/products/`, November 2010.

[9] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedings of the 12th USENIX Security Symposium*, pages 295–308. USENIX, Aug. 2003.

[10] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication application. In *Proceedings of the Symposium on VLSI Circuits*, pages 176–159, 2004.

[11] Y. Li, J. McCune, and A. Perrig. SBAP: Software-based attestation for peripherals. In A. Acquisti, S. Smith, and A.-R. Sadeghi, editors, *Trust and Trustworthy Computing*, volume 6101 of *Lecture Notes in Computer Science*, chapter 2, pages 16–29. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[12] J. Nick L. Petroni, T. Fraser, J. Molina, and W. A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the 13th USENIX Security Symposium* [25], pages 179–194.

[13] E. Öztürk, G. Hammouri, and B. Sunar. Towards Robust Low Cost Authentication for Pervasive Devices. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'08)*. IEEE Computer Society, March 2008.

[14] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297:2026–2030, 2002.

[15] B. Parno, J. M. McCune, and A. Perrig. Bootstrapping Trust in Commodity Computers. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 414–429, Oakland, CA, May 2010. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[16] U. Rührmair. SIMPL systems: On a public key variant of physical unclonable functions. Cryptology ePrint Archive, Report 2009/255, 2009.

[17] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. In *ACM CCS 2010*, 2010.

[18] A.-R. Sadeghi, C. Wachsmann, and I. Visconti. PUF-Enhanced RFID Security and Privacy. In *2nd Workshop on Secure Component and System Identification (SECSI 2010), Cologne, Germany, April 26-27, 2010*, April 2010.

[19] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *WiSe '06: Proceedings of the 5th ACM workshop on Wireless security*, pages 85–94, New York, NY, USA, 2006. ACM.

[20] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla. SWATT: SoftWare-based ATTestation for embedded devices. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 272–, Oakland, CA, May 2004. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[21] U. Shankar, M. Chew, and J. D. Tygar. Side effects are not sufficient to authenticate software. In *Proceedings of the 13th USENIX Security Symposium* [25], page 7.

[22] Trusted Computing Group (TCG). *TPM Main Specification, Version 1.2*, February 2005.

[23] Trusted Computing Group (TCG). *Mobile Trusted Module (MTM) Specifications*, May 2009.

[24] P. Tuyls and L. Batina. RFID-Tags for Anti-Counterfeiting. In *Proceedings of the Cryptographers' Track at the RSA Conference 2006 (CT-RSA'06)*, volume 3860 of *LNCS*, pages 115–131. Springer Verlag, February 2005.

[25] USENIX. *Proceedings of the 13th USENIX Security Symposium*, Berkeley, CA, USA, Aug. 2004.

[26] Verayo, Inc. Verayo website — product page. `http://www.verayo.com/product/products.html`, November 2010.

[27] Y. Yang, X. Wang, S. Zhu, and G. Cao. Distributed software-based attestation for node compromise detection in sensor networks. In *SRDS '07: Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, pages 219–230, Washington, DC, USA, 2007. IEEE Computer Society.