Technische Universität Darmstadt

Center for Advanced Security Research Darmstadt

Technical Report TR-2011-06-01

Lightweight Remote Attestation using Physical Functions

Steffen Schulz, Christian Wachsmann, Ahmad-Reza Sadeghi

System Security Lab
Technische Universität Darmstadt, Germany

Technische Universität Darmstadt
Center for Advanced Security Research Darmstadt
D-64293 Darmstadt, Germany

TR-2011-06-01
First Revision: June 3, 2011
Last Update: July 18, 2011

Lightweight Remote Attestation using Physical Functions*

Ahmad-Reza Sadeghi
TU Darmstadt (CASED)
& Fraunhofer SIT
Darmstadt, Germany
ahmad.sadeghi@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 standard software attestation, our scheme (i) is secure against collusion attacks to forge the attestation checksum, (ii) allows for the authentication and attestation of remote provers, and (iii) enables the detection of hardware attacks on the prover.

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*) [24]. 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 [7, 21] 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) [38] as a trust anchor for authentic reporting of a platform's software state. Today, TPMs are typically implemented as secure co-processors and available in many PCs, laptops, and server systems. The TCG also specifies the Mobile Trusted Module (MTM) [39], 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* [34] 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, 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 [15]. However, it has been shown that these side-effects are not appropriate to achieve a strong link to the underlying hardware [36, 19] as they only bind the software computation to *classes* of devices (i.e., of a specific type or manufacturer) instead of individual devices.

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 software attestation such that 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 collusions 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.

*This is the full version of [27].
Update: Dries Schellekens made us aware of a problem with the (optional, but useful) protocol for on-demand generation of CRPs (cf. Section 5.2). We also included a more detailed discussion of related work and security considerations.

Outline.

After providing background information on PUFs in Section 2, we present our new PUF-based attestation scheme in Section 3. We discuss different instantiations of our scheme in Section 4 and various optimizations for practical integration in Section 5. The security of our approach is analyzed in Section 6. Finally, we discuss related work in Section 7 and conclude in Section 8.

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 [23]. 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 [43, 13]. 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* [6], which can be efficiently implemented on resource-constrained devices [40]. Hence, PUFs can be used as deterministic functions.

Based on [2, 28], we consider PUFs that have the following properties, where `PUF` and $\texttt{PUF}'$ 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 $\texttt{PUF}'$ return different responses y and y'.
- *Pseudo-randomness:* It is infeasible to distinguish a `PUF` from a pseudo-random function.
- *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 PUF. 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 emulate 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 remote attestation.

The main protocol is the generalization of a typical software attestation protocol: The verifier $\mathcal{V}$ starts by sending a random challenge c 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 c, $\mathcal{P}$ sets up the initial checksum value σ_0 and some state r_0 as required by the underlying software attestation scheme. $\mathcal{P}$ then iteratively computes σ_k by taking k random measurement samples out of S. Specifically, in each iteration i of the checksum computation, $\mathcal{P}$ invokes three procedures: `GenMemAddr`(), `SwSum`(), and `HwSum`(). $\texttt{GenMemAddr}(r_{i-1}, y_{i-1})$ is used to generate an output r_i and a memory address a_i, which determines the next memory block b_i of S to be included into the software checksum as $\sigma_i \leftarrow \texttt{SwSum}(\sigma_{i-1}, b_i)$. 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-1}. 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 [34]. Hence, it is usually straightforward to

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

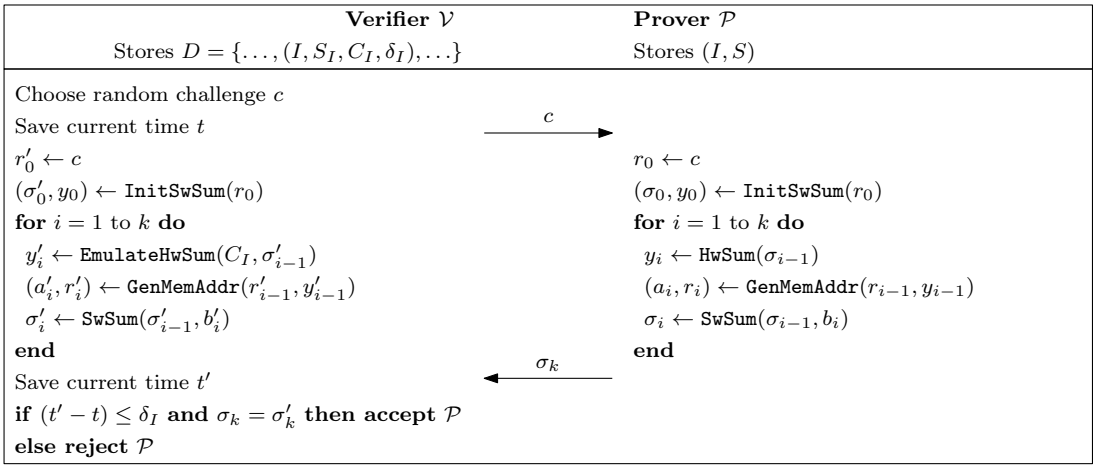

Figure 1: Remote attestation based on physical functions

integrate y_{i-1} 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 $y_i \leftarrow \texttt{HwSum}(\sigma_{i-1})$ to the `GenMemAddr`() procedure. As a result, every iteration of the software checksum additionally depends on the result of the device-specific 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}, such that `HwSum`() cannot be executed independently of `SwSum`(). Note that `HwSum`() and `SwSum`() are executed in parallel.

After every memory block b_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 emulating 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.

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.
- *Prover authentication:* A prover pretending to be another prover must be rejected by the verifier.
- *Prover isolation:* A prover colluding with other (malicious and/or more powerful) devices to forge the attestation 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 `GenMemAddr`() and the software checksum `SwSum`() components of our PUF-based attestation scheme can be instantiated based on any existing software-based attestation scheme (e.g., [31, 46, 4]) with only minor modifications to the underlying `GenMemAddr`() algorithm to integrate the hardware checksum `HwSum`(). In all modern software attestation designs, `GenMemAddr`() is implemented as a PRNG with internal state r_i that is used to generate pseudo-random memory addresses a_i. We can thus integrate the output y_{i-1} of `HwSum`() simply by merging it with the current state r_i in each iteration. Due to the unpredictability property of the PUF (Section 2), this is equivalent to (partly) re-seeding the PRNG, which effectively prevents the PRNG from repeating its sequence.

We use Secure Code Update By Attestation (SCUBA) [31] as an example to demonstrate the integration of existing software attestation schemes into our framework. We selected SCUBA since it targets resource-constrained devices

Algorithm 1: `GenMemAddr()` of SCUBA (modified).

Input:
r: current PRNG state
a: current memory address
$\underline{y}$: response of last hardware checksum query
$code_{\text{start}}, MASK$: memory address range to be measured
Output:
r: updated PRNG state
a: next memory address

`SCUBA-GenMemAddr(`r, a, y`)`
begin
  // T-function, updates r where $0 \leq r < 2^{16}$
  $r \leftarrow r + (r^2 \vee 5) \underline{\oplus y} \bmod 2^{16}$
  // Compute random memory address using r
  $a = ((a \oplus r) \wedge MASK) + code_{\text{start}}$
  return r, a
end

Algorithm 2: `SwSum()` of SCUBA.

Input:
σ: current checksum value
a: address of current memory block
i: loop counter (i.e., i-th iteration of `SwSum()`)
r: current PRNG state
Output: σ: updated checksum value
Data:
j: determines the part of the checksum to be modified
SR: current CPU status (flags) register value
PC: current CPU program counter value
`SCUBA-SwSum(`σ, a, i, r`)`
begin
  // Calculate checksum at index j
  // Note: $S[a]$ denotes block b of S at address a
  $\sigma_j \leftarrow \sigma_j + PC \oplus S[a] + i \oplus \sigma_{j-1} + r \oplus a + \sigma_{j-2} \oplus SR$
  $\sigma_j \leftarrow$`rotate-left(`σ_j`)`
  // Update checksum index ($10 \cdot 16 = 160$ bit)
  $j \leftarrow (j + 1) \bmod 10$
  return σ
end

and is presented with sufficiently detailed pseudo-code and experimental results.

Algorithm 1 depicts the (modified) `GenMemAddr()` algorithm of SCUBA, which implements a PRNG based on a T-function [16] that iterates over all possible PRNG states r. The output of the PRNG is used to generate a random memory address a in the range specified by the verifier. Algorithm 2 implements `SwSum()`, which is iteratively used to update the intermediate checksum value σ. This update is done by periodically iterating through the ten 16 bit parts of the 160 bits of σ. We modify Algorithm 1 such that the state update function of the PRNG (i.e., the T-function) integrates the output y of `HwSum()`, which makes the new PRNG state r dependant on y. Since the state update function iterates through all possible values of r, our modification introduces random jumps into the pseudo-random sequence, which does not weaken the security of the PRNG. note that similar modifications are possible with other PRNGs.

Based on Algorithm 1 and 2, the complete PUF-based attestation scheme is an instantiation of our high-level protocol design depicted in Figure 1.

4.2 Hardware Checksum

We present two alternative instantiations of the hardware checksum `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 `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 [22, 25]. 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 [22].

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 `PUF()`. Before deployment of $\mathcal{P}$, the interface for modelling the `PUF()` is then disabled such that any attempt to reactivate it leads to an irreversible change of `PUF()`. During deployment of $\mathcal{P}$, C_I and an algorithm `EmulateHwSum()` for emulating `HwSum()` is given to the verifier $\mathcal{V}$. In the attestation protocol, $\mathcal{P}$ computes `HwSum`$(\cdot)$ = `PUF`$(\cdot)$, whereas $\mathcal{V}$ emulates `HwSum`$(\cdot)$ = `EmulateHwSum`$(C_I, \cdot)$.

In practice, emulatable PUFs can be realized by most delay-based PUFs (e.g., Arbiter PUFs [18, 10] and Ring Oscillator PUFs [9]), which allow for creating precise mathematical models based on machine learning techniques [26]. 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 $\mathcal{V}$ can store only a subset of the CRP space. We thus have to deterministically limit the CRP subspace used during attestation without allowing $\mathcal{A}$ 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 `HwSum()`.

Commitment to procedure.

One approach is creating a database D of attestation challenge messages c 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 $\mathcal{P}$ in a secure environment. For each protocol run, $\mathcal{V}$ records in D the attestation challenge c 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 $(I, c, \sigma'_k, \delta_I) \in D$ and sends c to $\mathcal{P}$, which then computes σ_k using HwSum(). $\mathcal{V}$ accepts 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 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 possible inputs x_i to HwSum() generated by the intermediate checksum results σ_{i-1}, such that it becomes feasible to create a CRP database for node that is independent 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 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 a random attestation challenge r in the first message of the attestation protocol (Figure 1), i.e. $c = (q, r)$.

Specifically, we chose $f(\cdot)$ to be a function that maps intermediate checksum results σ_{i-1} to bit strings of length n and derive the input to HwSum() as $x_i \leftarrow q||f(\sigma_{i-1})$. Before deployment, the verifier $\mathcal{V}$ evaluates $y_j \leftarrow \texttt{HwSum}(q||j)$ for all $j \in \{0, \ldots, 2^n - 1\}$, and records $(q, y_0, \ldots, y_{2^n-1})$ as $C_{I,q}$ in D for a number of randomly chosen offsets q.

After deployment, $\mathcal{V}$ chooses a random $(I, S_I, C_{I,q}, \delta_I) \in D$ and a random nonce r to start the attestation. The prover $\mathcal{P}$ then computes the checksum σ_k using $\texttt{HwSum}(q||f(\sigma_{i-1}))$. While waiting for the response of $\mathcal{P}$, $\mathcal{V}$ computes the reference checksum σ'_k using $\texttt{EmulateHwSum}(C_{I,q}, q||f(\sigma'_{i-1}))$ 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[2] and would enable the adversary to pre-compute or even replay future attestation protocol runs.

5. PRACTICAL CONSIDERATIONS

In the following, we discuss some interesting variations of our attestation scheme to support its practical integration.

5.1 Checksum Synchronization

Our protocol assumes a strong inter-dependency and highly concurrent execution of the hardware and software checksum functions at the prover. However, in practice, the soft- and hardware checksum functions are limited by additional constraints such as cost, power consumption, and availability.[3]

[2]The encryption requires a secret key that must be protected against SW attacks. This typically requires a hardware security module, which, however, contradicts purpose of our lightweight attestation scheme.

[3]In particular, current PUF constructions are not designed or even evaluated for high access speed and we are currently investigating suitable PUF prototypes for the integration and practical evaluation of our protocol.

In the following we thus present two alternatives for a more abstract hardware checksum function $y_i \leftarrow \texttt{HwSum}'(x_i)$ as a wrapper to the actual hardware checksum HwSum(), which adapts the average access time of the hardware checksum to that of the software checksum and thus increases their inter-dependency.

In case the real hardware checksum HwSum() is faster than SwSum() and GenMemAddr(), we use a logical hardware function HwSum′() that simply issues $v > 1$ queries to the underlying HwSum() in a cipher-feedback mode, as illustrated in Algorithm 3. Note that Algorithm 3 enforces multiple subsequent queries to HwSum() to derive the final output of HwSum′() and may thus also be implemented in software. We include a counter u into the input of HwSum() for additional diffusion of its previous outputs y. However, several other diffusion functions are possible.

Algorithm 3: Low-speed hardware checksum HwSum′().

```
Input:
x: challenge to HwSum'()
v: number of iterations
Output: y: response of HwSum'()
Data: u: loop counter

HwSum'(x, v)
begin
    y ← 0
    for u = 0 to v do
        y ← y ⊕ HwSum(u||y ⊕ x)
    end
    return y
end
```

Algorithm 4: High-speed hardware checksum HwSum′().

```
Input: x: challenge to HwSum'()
Output: y: response of HwSum'()
Data:
v: PRF re-seeding period, fixed hardware parameter
u: loop counter, initialized as 0
seed: temporary PRF seed

HwSum'(x)
begin
    if u = 0 mod v then
        seed ← HwSum(x)
    end
    y ← PRF(seed, x)
    u ← u + 1
    return y
end
```

On the other hand, if the software checksum is faster than the hardware checksum, we require a construction where the actual HwSum() is queried less often than the software checksum. Unfortunately, this reduces the inter-dependency of the two algorithms and may allow an adversary to violate, e.g., prover isolation (Section 3). To mitigate this problem, we

propose the hardware checksum construction depicted in Algorithm 4, where a fast HwSum′() is implemented *in hardware*, using a fast PRF that is continuously re-seeded by HwSum(). If the seeding period is sufficiently low, this construction also remains secure against side-channel attacks since each PRF seed is only valid for a few microseconds. To achieve tamper-resistance, the implementation of HwSum′() must be protected as one of the security-critical device components, e.g., by physically shielding it with a coating PUF [41, 44].

Note that multiple HwSum() functions may integrated into the device, e.g., to provide comprehensive tamper-evidence for all hardware components through multiple PUFs. Moreover, they can be incorporated into Algorithm 4 to lower the frequency of accessing the individual HwSum() functions. Hereby, a global counter iterates over the number of available HwSum() functions and accesses them successively.

For example, the authors of [31] evaluate the SCUBA protocol for $k = 40,000$ and measure a legitimate attestation time frame of about 2.87 seconds. When instantiating our protocol based on SCUBA, the PUF should operate at $40,000/2.87 \approx 14$ kHz. Assuming that the PUF cannot operate at this frequency but only reaches, e.g., 200 Hz [5], and assuming that two PUFs are available per device $\mathcal{P}$, we can use Algorithm 4 and access the two PUFs in turns, resulting in an optimal PRF re-seeding period of about $v = 14 \text{ KHz}/(2 \cdot 200 \text{ Hz}) = 35$ or 2.5 milliseconds.

Finally, a special case of Algorithm 4 is where the Pseudo-Random Function (PRF) is seeded only once per attestation and the initial HwSum() challenge is a shared secret between prover and verifier (i.e., is stored in the prover's hardware to resist software compromise), which can be protected against hardware attacks using key storage based on PUFs [45, 20].

5.2 Key Establishment and CRP Generation

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 authentication secret and the proof of knowledge of this secret (the last message of the attestation protocol in Figure 1) is replaced with an authenticated key exchange.

However, contrary to the original proposal in [27] one cannot use σ_k as a session key. As pointed out by Dries Schellekens, an adversary can recover σ_k by recording the protocol messages exchanged between $\mathcal{P}$ and $\mathcal{V}$, then compromising the software of $\mathcal{P}$, and recompute σ_k by replaying the verifier's challenge c and simulating the attestation using HwSum() of $\mathcal{P}$ as an oracle. Knowing σ_k, the adversary can decrypt the required CRPs and future challenges c exchanged with the on-demand CRP generation protocol of [27]. The gained a priori knowledge on future CRPs compromises all future (remote) attestations.

To mitigate this attack, σ_k should be interpreted as an authentication secret that is used only in an authenticated key exchange that provides forward secrecy, such as authenticated Diffie-Hellman. Although this makes the solution less lightweight, we stress that the whole process of on-demand CRP generation itself can be rather resource intensive due to the exchange of the several thousand CRPs of HwSum().[4] Also note that sensor nodes are increasingly equipped with hardware acceleration for cryptography, such as many ZigBee-compatible wireless controllers, while secure memory in face of physical attacks is a real cost factor.

Moreover, more efficient solutions that a full Diffie-Hellman key exchange are possible. For example, the verifier $\mathcal{V}$ can generate an ephemeral RSA key pair with short public key exponent. The exponent and RSA modulus are transmitted to the verifier encrypted under σ_k. The prover $\mathcal{P}$ then can recover the public key and modulus to send an RSA-encrypted, randomly chosen session key K to $\mathcal{V}$. This scheme provides forward secrecy since the adversary needs the RSA secret key to recover K. This approach is similar to Encrypted Key Exchange [3], where a Diffie-Hellman public key is authenticated by encrypting it under a short user password[5].

5.3 Cooperative Attestation

As a last modification, we consider the problem of performing precise time measurements in the presence of network delays. In wireless sensor networks, hop-to-hop communication combined with energy restrictions at the sensor nodes used for routing can impose high transmission delays and more importantly jitter. Previous attempts to solve this issue are using trusted co-processors [17, 29]. However, we think a more practical and interesting solution is to leverage node collaboration as suggested in [46].

In particular, our protocol allows to delegate the time measurement to the direct neighbor of the prover without modification. The measuring neighbor only needs to record the response time and forwards it to the verifier in an authenticated fashion, together with the original response of the prover. Multiple neighbor nodes in the same broadcast zone can collaborate in this action and inform the base station about their measured time, resulting in a threshold-secure attestation scheme.

To attest the whole sensor network, this process starts at the verifier's direct neighbors and iteratively covers all nodes in the network. We must emphasize that the security of this approach also depends on the integrity of the neighbors of $\mathcal{P}$, i.e., the time it takes the adversary to compromise a given set of nodes after attestation. However, if the iterated attestation is executed as an attestation payload, the attack surface remains minimal.

6. SECURITY CONSIDERATIONS

In the following, we point out that our solution presented in Section 3 achieves the security goals of software attestation described in Section 3. Herby, we assume that secure software attestation schemes and PUFs exist (see Section 2).

Correctness and unforgeability.

Existing software attestation schemes consist of two main

[4] In contrast, the main attestation protocol requires the prover $\mathcal{P}$ to send only the final checksum response σ_k and the main cost for $\mathcal{P}$ is likely checksum computation itself, which by design must be computationally expensive.

[5] An information leakage attack is known against the encrypted Diffie-Hellamn key in the original scheme [3]. This *partition attack* can be mitigated by proper encoding and choice of the key RSA-key pair. Additionally, such an attack would be less effective in our case, where the encryption key σ_k is a strong secret used in only one session.

procedures, `GenMemAddr`() and `SwSum`(), which are iteratively executed to compute the checksum σ_k. Our proposed modifications to this algorithm are limited to the `GenMemAddr`() procedure, where we add the intermediate PUF responses y_{i-1} as additional randomizing input to the memory address computation. Hence, to guarantee that our changes do not affect correctness and unforgeability (see Section 3), we have to ensure that the output distribution of both the original and our modified `GenMemAddr`() procedure are computationally indistinguishable.

In practice, `GenMemAddr`() is typically implemented by a simple PRNG, which maintains an internal state that is (partly) used to generate pseudo-random outputs. The PUF response y_{i-1} must be incorporated into the PRNG such that all its subsequent outputs depend on y_{i-1} but without destroying their pseudo-randomness.

In case of the `GenMemAddr`() function of the SCUBA protocol (Algorithm 1), the PRNG is implemented as a T-function that, starting from a random seed, generates a pseudo-random sequence of 16 bit values. To incorporate the PUF response y_{i-1} into the PRNG, we can simply *add* y_{i-1} to the state of the T-function. Note that due to the unpredictability of the PUF (Section 2), y_{i-1} is a pseudo-random value. Hence, adding y_{i-1} to the state of the T-function is equivalent to reseeding the PRNG. However, if the PRNG is secure, insertion of additional entropy will not modify the statistical distribution of its output.

Prover identification.

The main security goal of our design is to link the checksum to the hardware of a prover $\mathcal{P}$. Our solutions achieves this by identifying $\mathcal{P}$ based on the outputs of its hardware checksum `HwSum`(). For this purpose, we must simply ensure that a sufficient amount of identifying information is generated by the hardware checksum `HwSum`() and incorporated into the attestation checksum σ_k to prevent simple guessing attacks, i.e., $k \cdot \texttt{len}(y_{i-1}) \geq 80$ bits, where k is the total number of `HwSum`() iterations and $\texttt{len}(y_{i-1})$ is the bit length of y_{i-1}.

Prover isolation.

The most challenging problem is to prevent the prover $\mathcal{P}$ from delegating (part of) the software checksum computation to other systems. In our design, we run the soft- and hardware checksum in parallel to create a large algorithmic interdependence between them. In the following, we derive constrains for the minimum amount of data that must be exchanged between hard- and software checksum to assure significant slowdowns in case of a delegation attack.

To detach the computation of the software checksum from the hardware of the prover, the adversary must simulate the hardware checksum output y_i in each iteration of the attestation algorithm to generate the correct input to the `GenMemAddr`() procedure, and the intermediate checksum values σ_i used as input to the hardware checksum. There are two major obstacles for the adversary to do this within the time frame allowed for successful attestation: (1) The adversary must involve the original hardware checksum procedure on the prover's hardware since this cannot be simulated or copied to another device due to the unpredictability and tamper-evidence (i.e., the unclonability) of the PUF. (2) The access speed of the hardware checksum is limited by its physical design and cannot be increased due to the tamper-evidence of the PUF.

Hence, an adversary that outsources the checksum computation to an external device is always limited by the maximum throughput of the prover's external communication interfaces when transferring the inputs and outputs x_i, y_i of the hardware checksum, which in turn are required in each iteration of the software checksum computation.

We assume a strong adversary that can calculate the simulated software checksum instantly. Such an adversary is only limited by the time t_{x_i} and t_{y_i} it takes to transmit x_i and y_i, respectively, plus the access time $t_{\texttt{HwSum}}$ of the hardware checksum: $t_{\mathcal{A}} = t_{x_i} + t_{y_i} + t_{\texttt{HwSum}}$. In contrast, the time for an unmodified checksum computation is limited by the computational speed of the legitimate hardware and software checksums, which are executed concurrently and thus have similar access time[6]: $t_{\mathcal{P}} = t_{\texttt{SwSum}} = t_{\texttt{HwSum}}$, where $t_{\texttt{SwSum}}$ is the access times of the software checksum. Hence, the security of our scheme depends on the amount of information that is transferred between hard- and software checksum during the computation of the checksum σ_k, and the time it takes to transfer this data over the accumulated transfer capacity of all external communication interfaces (ExtBps) of the prover:

$$t_{\mathcal{A}} - t_{\mathcal{P}} = t_{y_i} + t_{x_i} = \frac{k \cdot (\texttt{len}(x_i) + \texttt{len}(y_i))}{\text{ExtBps}}$$

In Section 4.2.2 we propose optimizations that work with reduced bit lengths for y_i and reduced entropy of x_i. The impact of reduced bit length of y_i is straightforward to assess by re-evaluation of the above security margins. However, the reduction of the challenge space changes may allow the adversary to predict the responses of the PUF in a similar way as the procedure used for on-demand generation of CRPs: After reception of the offset q, the compromised prover $\mathcal{P}$ is asked to generate the complete CRP subspace for the current attestation and to return only the responses, ordered by the corresponding challenges. In this case, the time $t_{\mathcal{A}}$ that the adversary takes to calculate the final software checksum value σ_k is limited only by the time required to generate and transmit all hardware checksum outputs y_i, i.e., $t_{\mathcal{A}} = 2^{\texttt{len}(x_i)} \cdot (t_{\texttt{HwSum}} + t_{y_i})$.The resulting timely advantage for the adversary can be derived as

$$t_{\mathcal{A}} - t_{\mathcal{P}} = \left(2^{\texttt{len}x_i} - k\right) t_{\texttt{HwSum}} + 2^{\texttt{len}(x_i)} t_{\texttt{SwSum}}$$

As we can see, the adversary's timely advantage critically depends on the length of x_i. Since we aim to keep $t_{y_i} = \texttt{len}(y_i)/\text{ExtBps}$ small to speed up on-demand generation and transmission of new CRP subsets and since $t_{\mathcal{A}}$ grows exponentially with increasing $\texttt{len}(x_i)$, we can set $\texttt{len}(y_i) = 1$, and use $\texttt{len}(x_i)$ as adjustable security parameter.

For instance, the Telos Mote used for the prototype implementation of SCUBA [31] supports a wireless interface with at most 250 kBps and a wired UART over USB interface with (typically) 115.2 kBps. Considering our example instantiation based on SCUBA for $\texttt{len}(x_i) = 20$ bits, the adversary would need to transmit $\texttt{len}(\sigma) + \texttt{len}(y_i) = 160 + 20$ bit for σ_i and y_i, respectively, in each iteration. The prototype implementation of [31] computes $40,000$ checksum iterations in less than 2.87 seconds. This requires the adversary to transmit $(160 + 20) \cdot 40,000 = 7,032$ kBit per attestation, incurring a significant overhead of $7,032$ kBit/(250 kBps+115.2 kBps) = 19.26 seconds.

[6]See Section 5.1 on how to synchronize the performance of hard- and software checksum.

Hence, if the implementation uses CRP sub-spaces of size $\texttt{len}(f(\sigma_i)) = 20$ bits and a `HwSum()` that operates at 14 kHz, an adversary, who aims to query the complete CRP subspace in a batch instead of executing software and hardware checksum in parallel, takes 74.85 seconds, which is a massive overhead compared to 2.85 seconds of the legitimate computation time.

7. RELATED WORK

The term *remote attestation* was coined by the TCG. It describes a process where a remote system reports its (software) state such that a manipulation of this state is noticed or no valid report can be generated at all.

Unfortunately, the mechanism suffers from the fundamental problem that software is only measured at load-time and runtime compromise is not detected. Furthermore, it requires a trusted component that is hard to compromise and thus typically expensive to implement in embedded systems such as sensor nodes or mobile devices.

The first conceptual description of assured remote code execution based on time constraints concerned the tamper-proof remote execution of mobile agents [11, 12]. Such agents are deployed with a signed expiration date that can be "recharged" via code obfuscation on a trusted platform. The idea was extended in [37, 15] to let the executed code securely report the software state and to assure that this state is running on actual hardware, i.e., is not simulated. Another proposal to link the software checksum calculation to hardware side-effects, such as the CPU state and caching effects that are supposedly expensive to simulate was made in [15]. Unfortunately, this scheme is vulnerable to several attacks [34, 36]: Redundancy in the code and simple hardware manipulations can be used to forge a valid attestation checksum within the required time. Furthermore, the proposed side-effects of the software execution are not reliable and not available on modern hardware. Finally, [36] presents multiple network-level attacks and shows that the scheme of [15] is generally unsuitable for access control, since it does not allow identification of the remote device or its user.

More recent works [33, 32, 35, 31, 46, 30, 19], extend software attestation by adapting the checksum algorithm to specific scenarios, such as sensor networks, and making it harder for the adversary to forge the checksum in time. However, purely software-based attestation in general always requires (1) a direct channel between verifier and prover to prevent man-in-the-middle attacks, (2) that the prover cannot communicate with colluding parties to delegate computation, and (3) that the hardware configuration is known and was not manipulated by the adversary. Specifically in case of sensor networks there is also the problem of network delay and jitter which may prevent accurate time measurements. However, this can be mitigated by delegating the time measurement to previously attested neighbors of the device to be attested [34, 46], or by using secure hardware [29]. Another approach to relax the requirements of software attestation is the exploitation of memory access time to increase the performance penalty for forged checksum computation [8, 14].

In contrast, our proposal is a generalization of [15, 29], and integrates an abstract hardware PRF into the computation of the software checksum to protect against collusion and hardware attacks.

8. CONCLUSION

We presented a novel approach to attest both the software and the hardware configuration of a remote embedded device that does not possess trusted hardware components. Our solution combines existing software attestation with cost-efficient physical security primitives, i.e., 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 devices to forge the software checksum. For future work we plan to evaluate our PUF-based attestation scheme using FPGA-based PUF prototypes.

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.

9. REFERENCES

[1] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Symposium on Research in Security and Privacy (S&P)*. IEEE, 1997.

[2] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In *Advances in Cryptology - ASIACRYPT*. Springer, 2009.

[3] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Symposium on Research in Security and Privacy (S&P)*. IEEE, 1992.

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

[5] P. F. Cortese, F. Gemmiti, B. Palazzi, M. Pizzonia, and M. Rimondini. Efficient and Practical Authentication of PUF-based RFID Tags. Technical Report RT-DIA-150-2009, Università degli studi Roma Tre, Dipartimento di Informatica e Automazione, 2009.

[6] 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*. Springer, 2004.

[7] 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.

[8] R. W. Gardner, S. Garera, and A. D. Rubin. Detecting code alteration by creating a temporary memory bottleneck. *Trans. Info. For. Sec.*, 4(4):638–650, 2009.

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

[10] B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas. Identification and authentication of integrated circuits: Research articles. *Concurrency and Computing: Practice & Experise*, 16(11):1077–1098, 2004.

[11] F. Hohl. An approach to solve the problem of malicious hosts. Technical report, University of Stuttgart, Faculty of Computer Science, Germany, 1997.

[12] F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*. Springer, 1998.

[13] Intrinsic ID. Intrinsic ID — product page. `www.intrinsic-id.com/products/`, 2010.

[14] M. Jakobsson and K.-A. Johansson. Retroactive Detection of Malware With Applications to Mobile Platforms. In *USENIX Workshop on Hot Topics in Security (HotSec'10)*. USENIX, 2010.

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

[16] A. Klimov and A. Shamir. New Cryptographic Primitives Based on Multiword T-Functions. In *Fast Software Encryption*. Springer, 2004.

[17] C. Krauss, F. Stumpf, and C. Eckert. Detecting node compromise in hybrid wireless sensor networks using attestation techniques. In *European conference on Security and privacy in ad-hoc and sensor networks (ESAS)*. Springer, 2007.

[18] 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 *Symposium on VLSI Circuits*, 2004.

[19] Y. Li, J. McCune, and A. Perrig. SBAP: Software-based attestation for peripherals. In *Trust and Trustworthy Computing*. Springer, 2010.

[20] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on VLSI Systems*, 13(10):1200–1205, 2005.

[21] J. Nick L. Petroni, T. Fraser, J. Molina, and W. A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *USENIX Security Symposium* [42].

[22] E. Öztürk, G. Hammouri, and B. Sunar. Towards Robust Low Cost Authentication for Pervasive Devices. In *International Conference on Pervasive Computing and Communications (PERCOM)*. IEEE, 2008.

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

[24] B. Parno, J. M. McCune, and A. Perrig. Bootstrapping Trust in Commodity Computers. In *Symposium on Research in Security and Privacy (S&P)*. IEEE, 2010.

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

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

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

[28] A.-R. Sadeghi, C. Wachsmann, and I. Visconti. PUF-enhanced RFID security and privacy. In *Workshop on Secure Component and System Identification (SECSI)*, 2010.

[29] D. Schellekens, B. Wyseur, and B. Preneel. Remote attestation on legacy operating systems with trusted platform modules. *Sci. Comput. Program.*, 74(1-2):13–22, 2008.

[30] A. Seshadri, M. Luk, and A. Perrig. SAKE: Software attestation for key establishment in sensor networks. *Distributed Computing in Sensor Systems*, pages 372–385, 2008.

[31] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *Workshop on Wireless security (WiSe)*. ACM, 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 *Symposium on Operating Systems Principles (SOSP)*. ACM, 2005.

[33] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Using SWATT for verifying embedded systems in cars. In *Embedded Security in Cars Workshop (ESCAR)*, 2004.

[34] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla. SWATT: SoftWare-based ATTestation for embedded devices. In *Symposium on Research in Security and Privacy (S&P)*. IEEE, 2004.

[35] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. Remote software-based attestation for wireless sensors. In *Security and Privacy in Ad-hoc and Sensor Networks*. Springer, 2005.

[36] U. Shankar, M. Chew, and J. D. Tygar. Side effects are not sufficient to authenticate software. In *USENIX Security Symposium* [42].

[37] D. Spinellis. Reflection as a mechanism for software integrity verification. *ACM Transactions on Information and System Security*, 3(1):51–62, 2000.

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

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

[40] P. Tuyls and L. Batina. RFID-tags for anti-counterfeiting. In *Cryptographers' Track at the RSA Conference (CT-RSA)*. Springer, 2006.

[41] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters. Read-proof hardware from protective coatings. In *Cryptographic Hardware and Embedded Systems Workshop*. Springer, 2006.

[42] USENIX. *Proceedings of the USENIX Security Symposium*, 2004.

[43] Verayo, Inc. Verayo website — product page. `www.verayo.com/product/products.html`, 2010.

[44] B. Škorić, S. Maubach, T. Kevenaar, and P. Tuyls. Information-theoretic analysis of capacitive physical unclonable functions. *Journal of Applied Physics*, 100(2):024902, 2006.

[45] B. Škorić, P. Tuyls, and W. Ophey. Robust key extraction from physical uncloneable functions. In *Applied Cryptography and Network Security (ACNS)*, 2005.

[46] 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)*. IEEE, 2007.