A Security Framework for the Analysis and Design of Software Attestation

Frederik Armknecht
Universität Mannheim,
Germany
armknecht@uni-mannheim.de

Ahmad-Reza Sadeghi
Technische Universität
Darmstadt (CASED),
Germany
ahmad.sadeghi@cased.de

Steffen Schulz[*]
Intel Corporation
steffen.schulz@intel.com

Christian Wachsmann
Intel CRI-SC at TU Darmstadt,
Germany
christian.wachsmann@cased.de

ABSTRACT

Software attestation has become a popular and challenging research topic at many established security conferences with an expected strong impact in practice. It aims at verifying the software integrity of (typically) resource-constrained embedded devices. However, for practical reasons, software attestation cannot rely on stored cryptographic secrets or dedicated trusted hardware. Instead, it exploits side-channel information, such as the time that the underlying device needs for a specific computation. As traditional cryptographic solutions and arguments are not applicable, novel approaches for the design and analysis are necessary. This is certainly one of the main reasons why the security goals, properties and underlying assumptions of existing software attestation schemes have been only vaguely discussed so far, limiting the confidence in their security claims. Thus, putting software attestation on a solid ground and having a founded approach for designing secure software attestation schemes is still an important open problem.

We provide the first steps towards closing this gap. Our first contribution is a security framework that formally captures security goals, attacker models and various system and design parameters. Moreover, we present a generic software attestation scheme that covers most existing schemes in the literature. Finally, we analyze its security within our framework, yielding sufficient conditions for provably secure software attestation schemes. We expect that such a consolidating work allows for a meaningful security analysis of existing schemes, supports the design of secure software attestation schemes and will inspire new research in this area.

1. INTRODUCTION

Embedded systems are increasingly permeating our information society, being more and more used also in security- and safety-critical applications. This generates an increasing need for enabling technologies that can validate and verify the integrity of a system's software state against malicious code. In this context, *software attestation* has become a popular research topic at many established security conferences with a large body of literature [15, 24, 26, 11, 23, 25, 22, 9, 21, 20, 3, 10, 14, 17, 19, 16, 29].

Software attestation is a trust establishment mechanism that allows a system, the *verifier*, to check the integrity of the program memory content of another system, the *prover*, against modification, e.g., by malicious code. As it mainly targets resource-constrained embedded systems (such as Atmel tinyAVR [2] microcontrollers), software attestation aims to work without any security hardware at the prover. Software attestation deploys the common approach of challenge-response protocols, where the verifier challenges the prover with respect to the expected memory content. However, cryptographic challenge-response protocols typically rely on secret values that should be unknown to malicious provers. This cannot be assumed for software attestation, where the provers are resource-constrained embedded systems that typically cannot afford secure hardware (such as a TPM) [28, 18, 27, 5, 16]. Hence, the adversary may get full control of the prover and its cryptographic secrets, rendering classical cryptographic primitives and protocols useless, a fact that demands for keyless security solutions.

Therefore software attestation follows a radically different approach than most conventional security mechanisms: It exploits the intrinsic physical constraints of the underlying hardware and side-channel information, typically the *computation time* required by the prover to complete the attestation protocol. More detailed, software attestation schemes are typically designed to temporarily utilize all the computing and memory resources of the prover, aiming at ensuring that the prover can only give the correct responses *in time* if the memory state is genuine. Of course this requires that the verifier can authenticate the device characteristics of the prover using an out-of-band channel, such as visual authentication, which clearly distinguishes software attestation from other approaches like remote attestation.

Without question, this requires completely different forms of security reasoning and likewise demands for other security assumptions on the underlying core functionalities and system properties, representing a highly challenging task. This may be the main reason that, despite its popularity and practical relevance, software attestation has not received any formal treatment yet but is still subject to ambiguities. To start with, there exist no common adversary model and no precise formalization of the security goals so far, hindering

[*] At the time of writing this author was affiliated with Ruhr-University Bochum, Germany, Macquarie University, Australia as well as the Intel Collaborative Research Institute for Secure Computing at TU Darmstadt, Germany.

a founded security analysis and making it difficult or even impossible to compare existing schemes.

Likewise the underlying security properties and assumptions have been only vaguely discussed, limiting the confidence in the security claims. In fact, current proposals often combine weak PRNGs and ad-hoc checksum function designs with unclear and possibly insufficient security properties. As a result, checksum collisions have been exploited directly to hide code modifications [24] and indirectly to manipulate the location of the measured code in the memory (memory copy attack [3]). Some works even propose to simply XOR consecutive memory blocks [32], leading to obvious collision attacks that were only indirectly considered in subsequent work [1]. Likewise, although several works consider the problem of free memory, i.e., unused sections of the memory, code compression attacks [3] have been ignored in recent works [17, 31] and considered as impractical [16] without giving any arguments.

Contribution.

In this paper, we make a first step towards putting software attestation on a solid ground. Our contributions are:

Security framework: We describe the first formal security framework for software attestation. This includes an adversary model that, interestingly, fundamentally deviates from classical cryptographic adversary models. Typically, the adversary is modelled by a *polynomially bounded* algorithm that aims to achieve a certain goal *without having certain knowledge* (e.g., cryptographic keys). In contrast, an adversary against a software attestation scheme can be *unbounded* in principle and has *complete knowledge* of the prover device configuration and state. However, during the attack it has to specify (or program) a malicious prover device with tight resource constraints. The goal is that this malicious prover can cheat in the attestation protocol within the *strict time bound* with *reasonable success probability* but *without* any interaction with the adversary. In other words, the adversary has unbound resources for *preparing* the attack but only a tight time-bound and limited computational and memory resources for *executing* the attack. Moreover we provide precise definitions for the security and the correctness of software attestation schemes and propose a formal system model that aims to balance between expressiveness and applicability. This allows a founded and comparable treatment of current and coming schemes and should help to avoid ambiguities in the future.

Generic software attestation scheme: We present a *generic* software attestation scheme that covers most existing software attestation protocols in the literature. Moreover, we identify and formalize several system parameters of software attestation and provide an upper bound of the success probability of a malicious prover against the generic scheme as a function of these parameters. The derived upper bound of the success probability implies *sufficient* conditions on the system parameters. Although some of these aspects have been implicitly assumed and informally discussed in the literature, we present their first formal treatment. Moreover, our approach provides new insights on how these parameters impact the security of the underlying software attestation scheme, which has never been analyzed before. This result allows to argue on the security of software attestation schemes by mapping the generic scheme and properties to the concrete scheme and by examining whether the properties are fulfilled. Moreover the generic scheme may serve as a blueprint for future schemes.

Argumentation techniques: The security treatment of the generic scheme required to use novel types of arguments. Since the common cryptographic proof technique of reducing the security of the scheme to a hard problem is not possible anymore, we had to argue *directly* that any attack strategy that is possible within the given time-bound fails with a certain probability. We conjecture that our approach may be of independent interest. For example, we expect that the security of concrete schemes that are not directly covered by the generic scheme may be argued using similar techniques.

New insights: Furthermore, our investigations yield new insights with respect to the cryptographic properties of the underlying primitives. Our work shows that cryptographic primitives can be used that are similar to established primitives, such as pseudo-random number generators or hash functions, but that differ in subtleties: Some cryptographic assumptions can be relaxed while others need to be strengthened. Such observations are relevant with respect to concrete realizations of secure software attestation schemes.

We see our work as a first step paving the way for a founded treatment of secure software attestation schemes. We expect that such a consolidating work allows for a meaningful security analysis of existing schemes and supports the design of arguably secure software attestation schemes and will inspire new research in this area.

Outline.

We give an overview of the related work in Section 2 and introduce our system model in Section 3. We present the formal framework for software attestation in Section 4, describe the generic software attestation scheme and its requirements in Section 5 and formally analyze its security in Section 6. Finally, we discuss our results and conclude in Section 7.

2. RELATED WORK

The existing literature on software attestation focuses on the design of checksum functions for different platform architectures and countering platform-specific attacks [24, 23, 9, 22, 17]. Several works consider the strengthening of checksum algorithms and their implementations against unintended modifications by either limiting the memory available to the prover during attestation [10, 29] or by using self-modifying and/or obfuscated attestation algorithms [25, 11]. Many works investigate the suitability and extension of software attestation to a variety of computing platforms, including sensors, peripherals and voting machines [21, 17, 9, 23, 16]. Furthermore, software attestation has been proposed as a key establishment mechanism [21].

Software attestation is different from *remote attestation* which has the goal to verify the integrity of remote provers, e.g., over a network. Specifically, remote attestation usually relies on secrets shared between the verifier and the honest prover, which is fundamentally different from software attestation that cannot rely on cryptographic secrets and instead typically assumes that the verifier can authenticate the prover using an out-of-band channel, such as visual authentication. Several works consider how to combine software attestation with hardware trust anchors such as TPMs and

SIM-cards [20, 16, 14] or intrinsic hardware characteristics such as code execution side-effects [15, 24, 26] and Physically Unclonable Functions [19]. Interestingly, most proposed implementations employ hash functions and PRNGs that are not cryptographically secure. Further, works that use cryptographically secure algorithms do not consider whether these algorithms maintain their security properties in the "keyless" software attestation scenario where the underlying secrets, such as the PRNG states, are known to the adversary. This is the reason why existing analysis papers on remote attestation, such as [4, 8], cannot be applied to software-based attestation, as they assume trusted hardware or software components. In this respect, our formal analysis provides a fundamental first step towards a deeper and more comprehensive understanding of software attestation.

An approach [13, 12] related to software attestation uses *Quines*. The basic idea is that the device outputs the whole content of its memory such that the verifier can compare it to the expected memory content. In contrast, software attestation aims to use short outputs only for practical reasons. In that sense, both approaches can be seen as special instantiations of proof-of-knowledge schemes where the proof either includes the knowledge itself (Quines) or responses depending on the knowledge (software attestation). A further difference is that, to reduce the impact of network jitter, software attestation typically minimizes the interaction between the prover and the verifier. In contrast the Quine-schemes in [13, 12] require significant interaction between the verifier and the device.

Similar to software attestation protocols, *proofs of work* schemes challenge the prover with computationally expensive or memory-bound tasks [6, 7]. However, while the goal of these schemes is to mitigate denial-of-service attacks and Spam by imposing artificial load on the service requester, the goal of software attestation schemes is using all of the prover's resources to prevent it from executing malicious code within a certain time frame. Hence, proofs of work are in general not suitable for software attestation since they are usually less efficient and not designed to achieve the optimality requirements of software attestation algorithms.

3. PRELIMINARIES

Notation. Let X, Y be two sets and $x \in X$ and $y \in Y$ be an element of each set. We denote with $f : X \rightarrow Y$ a function f that maps the set X to set Y. Further $f : x \mapsto y$ means that function f maps an element x to an element y. Let A and B be arbitrary algorithms. Then $y \leftarrow \mathsf{A}(x)$ means that on input x, A assigns its output to y. The expression A^{B} means that A has black-box access to B. We denote with $\mathsf{A}^{\widehat{\mathsf{B}}}$ an algorithm A that does *not* access an algorithm B. Let $\mathbb{D}$ be a probability distribution over the set X, then the term $x \overset{\mathbb{D}}{\leftarrow} X$ means the event of assigning an element of X to variable x that has been chosen according to $\mathbb{D}$. Further, we define $\mathbb{D}(x) := \Pr\left[x | x \overset{\mathbb{D}}{\leftarrow} X\right]$ for each $x \in X$ and denote with $\mathbb{U}$ the uniform distribution.

System Model. Software attestation is a protocol between a verifier $\mathcal{V}$ and a (potentially malicious) prover $\mathcal{P}$ where the latter belongs to a class of devices with clearly specified characteristics. That is, whenever we speak about a prover $\mathcal{P}$, we refer to a device that belongs to this class. Typically a prover $\mathcal{P}$ is a low-end embedded system that consists of memory and a computing engine (CE). The memory is composed of *primary memory* (PM), such as CPU registers and cache, and *secondary memory* (SM), such as RAM and Flash memory. We assume that the memory is divided into *memory words* and denote by $\Sigma := \{0,1\}^{l_s}$ the set of all possible memory words (e.g., $l_s = 8$ if words are bytes). Let $\mathfrak{s}$ and $\mathfrak{p}$ be the number of memory words that can be stored in SM and PM, respectively. An important notion is the state of a prover:

Definition 1 (State)**.** *Let $\mathcal{P}$ be a prover, i.e., a device that belongs to the specified class of devices. The* state $\mathsf{State}(\mathcal{P}) = S$ *of $\mathcal{P}$ are the memory words stored in the secondary memory (SM) of $\mathcal{P}$.*

Note that S includes the program code of $\mathcal{P}$ and hence specifies the algorithms executed by $\mathcal{P}$.

The computing engine (CE) comprises an arithmetics and logic unit that can perform computations on the data in primary memory (PM) and alter the program flow. For performance reasons, PM is typically fast but also expensive. Hence, the size of PM is usually much smaller than the size of SM. To make use of SM, CE includes the Read instruction to transfer data from SM to PM and the Write instruction to write data from PM to SM. More precisely, $\mathsf{Read}(S, a, b)$ takes as input a memory address a of SM and a memory address b of PM and copies the data word x stored at address a in SM to the data word at address b in PM. For convenience, we write $\mathsf{Read}(S, a)$ instead of $\mathsf{Read}(S, a, b)$ whenever the address b of PM is not relevant. Note that $\mathsf{Read}(S, a, b)$ overwrites the content y of PM at address b. Hence, in case y should not be lost, it must be first copied to SM using Write or copied to another location in PM before Read is performed. It is important to stress that, whenever CE should perform some computation on some value x stored in SM, it is mandatory that x is copied to PM before CE can perform the computation. Further, since SM is typically much slower than PM, Read and Write incur a certain time overhead and delay computations on x. We denote the time required by CE to perform some instruction or algorithm Ins with $\mathsf{Time}(\mathsf{Ins})$. Note that we only consider provers as described above while the verifier $\mathcal{V}$ can be an arbitrary computing platform that may interact with $\mathcal{P}$.

Remark 1: Platform Architecture. Note that we focus on embedded microcontrollers since these are commonly targeted by software attestation. We explicitly exclude provers that are high-end computing platforms with multiple CPUs and/or Direct Memory Access (DMA) since these are typically equipped with secure hardware (such as TPMs) and hence could support common cryptographic solutions based on secrets. Further, their memory architectures are usually more complex than in our system model. In particular, such platforms usually feature additional hardware to predict and fetch memory blocks in advance, making the time-bounded approach much more difficult and its realization highly dependent on the concrete system.

4. SECURE SOFTWARE ATTESTATION

Secure software attestation enables the verifier $\mathcal{V}$ to gain assurance that the state of a prover $\mathcal{P}$ is equal to a particular state S. If this is the case, we say that $\mathcal{P}$ is in state S, i.e., formally $\mathsf{State}(\mathcal{P}) = S$. Consequently, a prover $\mathcal{P}$ is called *honest* (with respect to some state S) if $\mathsf{State}(\mathcal{P}) = S$, otherwise it is considered to be *malicious.*

Remark 2: Distance between Honest and Malicious Prover State. Observe that a prover $\widetilde{\mathcal{P}}$ is already considered to be malicious even if its state differs by only one state entry (memory word) from S. This is a necessary consequence of the goal of having a definition of honest and malicious provers that is as generic as possible.

The common approach of software attestation is to validate the prover's state by requesting N random samples of the prover memory content. Hence, the Hamming distance λ between the state S of an honest prover and the state $\widetilde{S}$ of a malicious prover $\widetilde{\mathcal{P}}$ directly affects a malicious prover's success probability. As far as we know, we are the first to formally take into account the impact of λ on the security of software attestation schemes (cf. λ in Theorem 1).

Ideal Approach. Ideally, $\mathcal{V}$ could disable the computing engine (CE) of $\mathcal{P}$ and directly read and verify the state S stored in the secondary memory (SM) of $\mathcal{P}$. However, exposing CE and SM of $\mathcal{P}$ to $\mathcal{V}$ in such a way requires hardware extensions[1] on $\mathcal{P}$, which contradicts the goal of software attestation to work with no hardware modifications.

Practical Approach. As the ideal approach is not feasible in practice, the common approach in the literature is that $\mathcal{V}$ and $\mathcal{P}$ engage in a *challenge-response protocol* Attest where $\mathcal{P}$ must answer to a challenge of $\mathcal{V}$ with a response that depends on S. In the following, whenever we refer to a software attestation scheme we actually mean the corresponding challenge-response protocol Attest. Observe that Attest needs to include a description of the algorithm that processes the verifier's challenge and computes the prover's response.

In general, software attestation aims to figure out whether the original state S of a device has been replaced by the adversary with a malicious state $\widetilde{S} \neq S$. Observe that although $\widetilde{S}$ is different from S, we cannot exclude that $\widetilde{S}$ may depend on S. This implies an important difference to common cryptographic scenarios: Software attestation cannot rely on any secrets since the adversary has access to the same information as the honest prover $\mathcal{P}$. Therefore software attestation follows a fundamentally different approach and leverages side-channel information, typically the time δ the prover takes to compute the response. A basic requirement of this approach is that S specifies a practically optimal implementation of the algorithm that processes the challenge according to Attest. This means that it should be hard to find any other implementation of this algorithm that can be executed by a prover $\mathcal{P}$ in significantly less time than δ. Otherwise, a malicious prover could use a faster implementation and exploit the time difference to perform additional computations, e.g., to lie about its state.

Furthermore, the communication time jitter between $\mathcal{V}$ and $\mathcal{P}$ is typically much higher than the time needed by the computing engine of $\mathcal{P}$ to perform a few instructions. Hence, to ensure that $\mathcal{V}$ can measure also slight changes to the prover's code (that could be exploited by a malicious prover to lie about its state), $\mathcal{V}$ needs to amplify the effect of such changes. The most promising approach to realize this in practice is designing the attestation protocol as an iterative algorithm with a large number of rounds.

Further, since showing the optimality of complex implementations is a hard problem and since $\mathcal{P}$ must compute the response in a reasonable amount of time, it is paramount that the individual rounds are simple and efficient. As a result, cryptographically secure hash functions and complex Pseudo-Random Number Generators (PRNGs) are not a viable option. Hence, several previous works deployed lightweight ad-hoc designs of compression functions and PRNGs, however, without analyzing the underlying requirements on these components and their interaction. In contrast, we identify concrete sufficient requirements.

Adversary Model and Security Definition.

In the following, we provide the first formal specification of the adversary model and the security of a software attestation scheme Attest based on a security experiment $\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}$ that involves an adversary $\mathcal{A}$. The experiment is divided into two phases and works as follows:

Preparation Phase: At the beginning, the adversary $\mathcal{A}$ receives as input a state S and a time-bound δ. It outputs a (possibly) malicious prover $\widetilde{\mathcal{P}}$ by specifying its state $\widetilde{S}$, i.e., $\mathsf{State}(\widetilde{\mathcal{P}}) = \widetilde{S}$.

Execution Phase: The prover $\widetilde{\mathcal{P}}$ specified in the previous phase receives a challenge c and returns a "guess" $\widetilde{r}$ for the correct response r.

The result of the experiment is ACCEPT if $\widetilde{\mathcal{P}}$ responded within time δ and $\widetilde{r} = r$, and REJECT otherwise.

Based on this experiment we define *correctness* and *soundness.* Correctness is defined analogously to the common meaning of correctness of challenge-response protocols: In case $\mathsf{State}(\widetilde{\mathcal{P}}) = S$, that is $\widetilde{\mathcal{P}}$ is in the state expected by $\mathcal{V}$, the prover $\widetilde{\mathcal{P}}$ should always succeed, i.e., the result of the experiment should always be ACCEPT. Soundness means that in case $\mathsf{State}(\widetilde{\mathcal{P}}) \neq S$, the probability that the result of the experiment is ACCEPT should be below a certain threshold.

Definition 2 (Correctness and Soundness)**.** *Consider a software attestation scheme* Attest *and a state S. For a given adversary $\mathcal{A}$ we denote by* `EqualState` *the event that the output of $\mathcal{A}$ during the experiment is a prover $\widetilde{\mathcal{P}}$ with* $\mathsf{State}(\widetilde{\mathcal{P}}) = S$.

The software attestation scheme Attest *is* correct *if for all adversaries $\mathcal{A}$ it holds that*

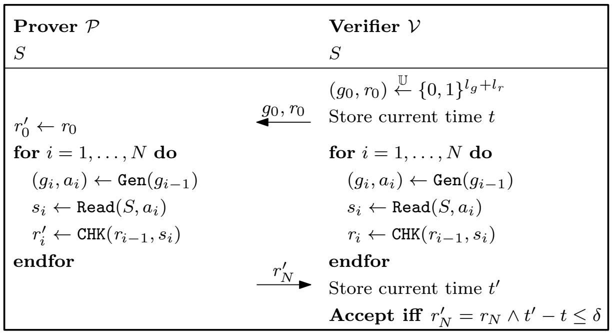
$$\Pr\left[\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}(S) = \text{ACCEPT} \middle| \mathtt{EqualState}\right] = 1.$$

Attest *is* ε-secure *if for all adversaries $\mathcal{A}$ it holds that*

$$\Pr\left[\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}(S) = \text{ACCEPT} \middle| \neg\mathtt{EqualState}\right] \leq \varepsilon.$$

Remark 3: Power of $\mathcal{A}$. The security of software attention significantly differs from common cryptographic models, where the time effort of the adversary is typically bounded (often polynomially bounded in some security parameter). More detailed, in the preparation phase, $\mathcal{A}$ can be any *unrestricted* probabilistic algorithm. However, $\mathcal{A}$ has no influence anymore once Attest is executed between $\widetilde{\mathcal{P}}$ and $\mathcal{V}$ in the execution phase. As $\widetilde{\mathcal{P}}$ is a device with the same

[1] Note that existing testing interfaces such as JTAG cannot be used since they are typically disabled on consumer devices to prevent damage to the device and unintended reverse-engineering.



```
Prover $\mathcal{P}$                              Verifier $\mathcal{V}$
$S$                                               $S$

                                                  $(g_0, r_0) \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_g+l_r}$
                              $g_0, r_0$ ←        Store current time $t$
$r'_0 \leftarrow r_0$
for $i = 1, \ldots, N$ do                         for $i = 1, \ldots, N$ do
    $(g_i, a_i) \leftarrow \texttt{Gen}(g_{i-1})$     $(g_i, a_i) \leftarrow \texttt{Gen}(g_{i-1})$
    $s_i \leftarrow \texttt{Read}(S, a_i)$            $s_i \leftarrow \texttt{Read}(S, a_i)$
    $r'_i \leftarrow \texttt{CHK}(r_{i-1}, s_i)$      $r_i \leftarrow \texttt{CHK}(r_{i-1}, s_i)$
endfor                                            endfor
                              $r'_N$ →            Store current time $t'$
                                                  Accept iff $r'_N = r_N \wedge t' - t \leq \delta$
```

Figure 1: The Generic Attestation Scheme Attest

characteristics as an honest prover, $\widetilde{\mathcal{P}}$ has to comply to the same restrictions as $\mathcal{P}$. In other words, the adversary has unbounded resources for *preparing* the attack but only a tight time-bound and limited resources for *executing* the attack.

Observe that this reflects the strongest possible adversary model, which in principle could be relaxed by imposing bounds during the preparation phase.

Remark 4: Difference to Remote Attestation. The goal of *remote attestation* is to verify the integrity of remote provers, e.g., over a network. In particular, in practice a verifier $\mathcal{V}$ usually cannot exclude that a malicous prover may have more computational power than the honest prover. Therefore, remote attestation schemes usually rely on secrets shared between the verifier and the honest prover.

This is fundamentally different from software attestation which cannot rely on cryptographic secrets to authenticate the prover device to $\mathcal{V}$. Hence, as already elaborated, existing works on software attestation typically assume that $\mathcal{V}$ can authenticate the prover hardware using an out-of-band channel, such as visual authentication.

5. GENERIC SOFTWARE ATTESTATION

In this section, we formalize a generic software attestation scheme that captures most existing schemes in the literature. In particular, we formally define several aspects and assumptions, most of them being only informally discussed or implicitly defined so far.

5.1 Protocol Specification

The main components of our generic attestation scheme (cf. Figure 1) are two deterministic algorithms:

- Memory address generator:
  $\mathsf{Gen} : \{0,1\}^{l_g} \to \{0,1\}^{l_g} \times \{0,1\}^{l_a}$, $g \mapsto (g', a')$
- Compression function:
  $\mathsf{Chk} : \{0,1\}^{l_r} \times \Sigma \to \{0,1\}^{l_r}$, $(r, s) \mapsto r'$

Here l_g, l_a and l_r are the bit length of the state g of Gen, the memory addresses a and the attestation response r', respectively, and Σ is the set of possible state entries (memory words). Both algorithms are iteratively applied within the scheme over $N \in \mathbb{N}$ rounds. For the sake of readability, we provide an iterative definition of Chk^N: For some $r_0 \in \{0,1\}^{l_r}$ and $\vec{s} := (s_1, \ldots, s_N)$, we define $r_N \leftarrow \mathsf{Chk}^N(c, \vec{s})$ as $r_i := \mathsf{Chk}(r_{i-1}, s_i)$ for $i = 1, \ldots, N$.

The protocol works as follows: The verifier $\mathcal{V}$ sends a random attestation challenge (g_0, r_0) to the prover $\mathcal{P}$, which is used as seed for Gen and Chk. Specifically, $\mathcal{P}$ iteratively generates a sequence of memory addresses $(a_1, \ldots, a_N)$ based on g_0 using Gen. For each $i \in \{1, \ldots, N\}$, $\mathcal{P}$ reads the state entry $s_i = \mathsf{Read}(S, a_i)$ at address a_i and iteratively computes $r'_i = \mathsf{Chk}(r'_{i-1}, s_i)$ using $r'_0 = r_0$. Finally, $\mathcal{P}$ sends r'_N to $\mathcal{V}$, which executes exactly the same computations[2] as $\mathcal{P}$ using the state S and compares the final result with the response r'_N from $\mathcal{P}$. Eventually, $\mathcal{V}$ accepts iff $r'_N = r_N$ and $\mathcal{P}$ responded in time $\leq \delta := N(\delta_{\mathsf{Gen}} + \delta_{\mathsf{Read}} + \delta_{\mathsf{Chk}})$, where δ_{Gen}, δ_{Read} and δ_{Chk} are upper time-bounds for running Gen, Read and Chk, respectively, on a genuine and honest prover.

In practice the delay for submitting and receiving messages needs to be considered. The common approach is to choose N, the number of rounds, big enough such that this delay is small compared to the runtime of the protocol. For simplicity, we assume that this is the case in the following and hence ignore the time for sending messages.

Remark 5: Correctness. Observe that an honest prover $\mathcal{P}$ always makes an honest verifier $\mathcal{V}$ accept since both perform exactly the same computations on the same inputs and the honest prover by assumption requires at most time δ.

Remark 6: Generality of the Protocol. Note that the basic concept of our generic scheme and several instantiations of this concept for specific platforms can be found in the literature on software attestation (cf. Section 2). However, we aim at abstracting from the particularities of individual platforms and instead design and analyze a construction that is as generic as possible. Further, some existing software attestation schemes also use the memory addresses a_i and/or the index i as input to the checksum function Chk. However, since there is a clear dependence between the index i, the memory address a_i and the memory block $s_i = \mathsf{Read}(S, a_i)$ and since the use of simple components is a primary goal of software attestation, we restrict to the case where only the memory blocks are used as input to the checksum function.

5.2 Design Criteria and Properties

Next, we discuss the design criteria of the underlying algorithms and formally define their properties required later in the security analysis. Note that, although some of these properties have been informally discussed or implicitly made in prior work, they have never been formally specified and analyzed before.

5.2.1 Implementation of the Core Functionalities

The generic protocol deploys three core functionalities: Read, Gen and Chk, which of the execution time is of paramount importance for the security of software attestation. Hence, we make the following assumptions that are strongly dependent on the concrete implementation and prover hardware and are hard to cover in a generic formal framework:

1. **Optimality:** There is no implementation of Read, Gen and Chk (or their combination) that is more efficient (with respect to time and/or memory) than the implementation used by the honest prover in state S.

[2] Note that the verifier could also compute the expected result before, after or in parallel to the protocol.

2. **Atomicity:** It is not possible to execute Read, Gen and Chk only partially, e.g., by omitting some of the underlying instructions.

We formally cover these assumptions by modelling Read, Gen and Chk as oracles. That is, whenever $\mathcal{P}$ wants, e.g., to execute $\mathsf{Read}(\mathsf{State}(\mathcal{P}), a)$, $\mathcal{P}$ sends a to the Read-oracle and receives the corresponding result s. While sending and receiving messages between $\mathcal{P}$ and the oracles are modelled to take no time, the *determination* of the response does. More precisely when $\mathcal{P}$ invokes one of these oracles, it takes a certain amount of time before $\mathcal{P}$ gets the result. Within this time period $\mathcal{P}$ is inactive and cannot perform any computations. We denote the response time of the Read, Gen and Chk-oracle by δ_{Read}, δ_{Gen} and δ_{Chk}, respectively. Moreover the inputs to and the outputs of the oracles need to be stored in the primary memory of $\mathcal{P}$.

Remark 7: Order of Computations. A consequence of this modelling approach is that a malicious prover $\widetilde{\mathcal{P}}$ can compute the outputs of Gen and Chk only in the correct order. For instance, before $\widetilde{\mathcal{P}}$ can determine s_i it must first determine s_{i-1}. Given that concrete instantiations of the generic scheme are iteratively executed, the limited size of the primary memory (PM) (see below) and the fact that accessing the secondary memory requires significantly more time than accessing PM, we consider this assumption to be reasonable for most practical instantiations.

5.2.2 System-Level Properties

The size and utilization of the primary memory (PM) plays a fundamental role for assessing the optimality of a software attestation scheme with regard to the resources used by a prover $\widetilde{\mathcal{P}}$. Therefore, a common assumption is that PM is just big enough to honestly execute Attest, i.e., there is no free PM that could be used otherwise.[3]

Another crucial assumption of any software attestation scheme not explicitly made in most previous works is that the state S should not be compressible into PM. For instance, consider the extreme case where all entries of S contain the same value s. In this case a malicious prover $\widetilde{\mathcal{P}}$ could easily determine the correct attestation response by simply storing s in PM while having a different state $\mathsf{State}(\widetilde{\mathcal{P}}) \neq S$. Hence, we require that $\widetilde{\mathcal{P}}$ should not be able to determine a randomly selected state entry s_i of S without accessing the secondary memory with better probability than guessing:

Definition 3 (State Incompressibility)**.** *For a state S, let $\mathbb{D}_S$ denote the probability distribution of S in the following sense: For any state entry $x \in \Sigma$ it holds that*

$$\mathbb{D}_S(x) := \Pr\left[x = s | a \xleftarrow{\mathbb{U}} \{0,1\}^{l_a} \wedge s := \mathsf{Read}(S, a)\right].$$

S is called incompressible *if for any algorithm $\mathsf{Alg}^{\widehat{\mathsf{Read}}}$ that can be executed by the prover $\mathcal{P}$ and that does not invoke Read, it holds that*

$$\Pr\left[\widetilde{s} = s | a \xleftarrow{\mathbb{U}} \{0,1\}^{l_a} \wedge s = \mathsf{Read}(S, a) \wedge \widetilde{s} \leftarrow \mathsf{Alg}^{\widehat{\mathsf{Read}}}(a) \right.$$
$$\left. \wedge\, \mathsf{Time}_{\mathcal{P}}(\mathsf{Alg}^{\widehat{\mathsf{Read}}}) \leq \delta_{\mathsf{Read}}\right] \leq \gamma = \max_{x \in \Sigma} \mathbb{D}_S(x).$$

5.2.3 Cryptographic Properties

Although it is quite obvious that the security of the software attestation scheme depends on the cryptographic properties of Gen and Chk, these requirements have not been systematically analyzed and formally specified before. While it would be straightforward to model these functions as pseudorandom number generators (PRNGs) and hash functions (or even random oracles), respectively, there are some subtle differences to the common cryptographic scenario which must be carefully considered. As we elaborate below, Gen needs to meet a property which is *stronger* than the common security definition of cryptographic PRNGs while for Chk a significantly *weaker* condition than the classical security properties of hash functions is sufficient.

Pseudo-Randomness of the Outputs of Gen. To prevent a malicious prover $\widetilde{\mathcal{P}}$ from using pre-computed attestation responses, the memory addresses a_i generated by Gen should be "sufficiently random". Ideally, all combinations should be possible for $(a_1, \ldots, a_N)$. While this is impossible from an information-theoretic point of view, the best one may ask for is that the memory addresses a_i generated by Gen should be computationally indistinguishable from uniformly random values within a certain time-bound t:

Definition 4 (Time-Bounded Pseudo-Randomness of Gen)**.** $\mathsf{Gen} : \{0,1\}^{l_g} \to \{0,1\}^{l_g + l_a}$ *is called* (t, ϱ)-pseudo-random *if for any algorithm Alg that can be executed by $\mathcal{P}$ in $\mathsf{Time}(\mathsf{Alg}) \leq t$ it holds that*

$$\left| \Pr\left[b = 1 | g_0 \xleftarrow{\mathbb{U}} \{0,1\}^{l_g} \right.\right.$$
$$\wedge\, (g_i, a_i) \leftarrow \mathsf{Gen}(g_{i-1}) : i \in \{1, \ldots, N\}$$
$$\left. \wedge\, b \leftarrow \mathsf{Alg}(a_1, \ldots, a_N)\right]$$
$$- \Pr\left[b = 1 | a_i \xleftarrow{\mathbb{U}} \{0,1\}^{l_a} : i \in \{1, \ldots, N\} \right.$$
$$\left.\left. \wedge\, b \leftarrow \mathsf{Alg}(a_1, \ldots, a_N)\right] \right| \leq \varrho.$$

Observe that this definition requires that Alg does not know the seed g_0 of Gen, which is *not* given in the generic software attestation scheme. In principle nothing prevents a malicious prover $\widetilde{\mathcal{P}}$ from using g_0 to compute the addresses $(a_1, \ldots, a_N)$ on its own, making them easily distinguishable from random values. The best we can do is to require that $\widetilde{\mathcal{P}}$ cannot derive any meaningful information about a_{i+1} from g_i without investing a certain minimum amount of time. Specifically, we assume that an algorithm with input g that does *not* execute Gen cannot distinguish $(g', a') = \mathsf{Gen}(g)$ from uniformly random values. Formally:

Definition 5 (Time-Bounded Unpredictability of Gen)**.** $\mathsf{Gen} : \{0,1\}^{l_g} \to \{0,1\}^{l_g} \times \{0,1\}^{l_a}$ *is* ν_{Gen}-unpredictable *if for any algorithm $\mathsf{Alg}^{\widehat{\mathsf{Gen}}}$ that can be executed by $\mathcal{P}$ and that does not*

[3] Possible measures to achieve this are either to choose Gen and/or Chk accordingly or to execute Gen and/or Chk several times per round on different inputs to occupy more primary memory.

execute Gen*, it holds that*

$$\Big| \Pr\Big[b = 1 | g \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_g} \wedge (g', a') \leftarrow \mathsf{Gen}(g) \\ \wedge\, b \leftarrow \mathsf{Alg}^{\widehat{\mathsf{Gen}}}(g, g', a')\Big] \\ - \Pr\Big[b = 1 | g \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_g} \wedge (g', a') \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_g} \times \{0,1\}^{l_a} \\ \wedge\, b \leftarrow \mathsf{Alg}^{\widehat{\mathsf{Gen}}}(g, g', a')\Big]\Big| \leq \nu_{\mathsf{Gen}}.$$

Weakened Pre-image Resistance of Chk**.** The purpose of the compression function Chk^N is to map the state S of the prover $\mathcal{P}$ to a smaller attestation response r_N, which reduces the amount of data to be sent from $\mathcal{P}$ to the verifier $\mathcal{V}$. Observe that the output of Chk^N depends also on the challenge sent by the verifier to avoid simple replay attacks and the pre-computation of attestation responses. A necessary security requirement on Chk is that it should be hard for a malicious prover $\widetilde{\mathcal{P}}$ to replace the correct input $\vec{s} = (s_1, \ldots, s_N)$ to Chk with some other value $\vec{\tilde{s}} \neq \vec{s}$ that yields the same attestation response r_N as $\vec{s}$. This is similar to the common notion of *second pre-image resistance* of cryptographic hash functions. However, due to the time-bound of the software attestation scheme it is sufficient that Chk^N fulfills only a much weaker form of second pre-image resistance since we need to consider only "blind" adversaries who (in contrast to the classical definition of second pre-image resistance) do not know the correct response r_N to the verifier's challenge (g_0, r_0). The reason is that, as soon as $\mathcal{P}$ knows the correct response r_N, he could send it to $\mathcal{V}$ and would not bother to determine a second pre-image. Hence, we introduce the definition of blind second pre-image resistance which concerns algorithms that are given only part of the input $\vec{s}$ of Chk^N and that have to determine the correct output of $\mathsf{Chk}^N(r_0, \vec{s})$:

Definition 6 (Blind Second Pre-image Resistance)**.** $\mathsf{Chk} : \{0,1\}^{l_r} \times \Sigma \rightarrow \{0,1\}^{l_r}$ *is* ω-blind second pre-image resistant *with respect to the distribution* $\mathbb{D}_S$ *(cf. Definition 3) if for any* $N \in \mathbb{N}$*, any subset of indices* $J \subsetneq \{1, \ldots, N\}$ *and for any algorithm* Alg *that can be executed by* $\mathcal{P}$*, it holds that*

$$\Pr\Big[\tilde{r} = r | r_0 \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_r} \wedge s_i \overset{\mathbb{D}_S}{\leftarrow} \Sigma : i \in \{1, \ldots, N\} \\ \wedge\, \tilde{r} \leftarrow \mathsf{Alg}(r_0, (s_j)_{j \in J}) \wedge r \leftarrow \mathsf{Chk}^N(r_0, s_1, \ldots, s_N)\Big] \leq \omega.$$

In addition we require (similar to Definition 5) that $\widetilde{\mathcal{P}}$ cannot determine any information on $r_N = \mathsf{Chk}^N(r_0, s_1, \ldots, s_N)$ without executing Chk^N:

Definition 7 (Unpredictability of Chk^N)**.** $\mathsf{Chk} : \{0,1\}^{l_r} \times \Sigma \rightarrow \{0,1\}^{l_r}$ *is* ν_{Chk}*-unpredictable with respect to the distribution* $\mathbb{D}_S$ *(cf. Definition 3) if for any algorithm* $\mathsf{Alg}^{\widehat{\mathsf{Chk}^N}}$ *that can be executed by* $\mathcal{P}$ *and that does not execute* Chk^N*, it holds that*

$$\Big| \Pr\Big[b = 1 | r_0 \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_r} \wedge s_i \overset{\mathbb{D}_S}{\leftarrow} \Sigma : i \in \{1, \ldots, N\} \\ \wedge\, r = \mathsf{Chk}^N(r_0, s_1, \ldots, s_N) \\ \wedge\, b \leftarrow \mathsf{Alg}^{\widehat{\mathsf{Chk}^N}}(r_0, s_1, \ldots, s_N, r)\Big] \\ - \Pr\Big[b = 1 | r_0 \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_r} \wedge s_i \overset{\mathbb{D}_S}{\leftarrow} \Sigma : i \in \{1, \ldots, N\} \\ \wedge\, r \overset{\mathbb{U}}{\leftarrow} \{0,1\}^{l_r} \\ \wedge\, b \leftarrow \mathsf{Alg}^{\widehat{\mathsf{Chk}^N}}(r_0, s_1, \ldots, s_N, r)\Big]\Big| \leq \nu_{\mathsf{Chk}}.$$

6. SECURITY OF THE SCHEME

In this section we derive an upper bound for the success probability of a malicious prover $\widetilde{\mathcal{P}}$ to make the verifier $\mathcal{V}$ accept. This bound depends on the parameters defined in Section 5.2 which provide a sufficient condition to prove the generic attestation scheme secure. The bound is as follows:

Theorem 1 (Generic Upper Bound)**.** *Let* S *be an incompressible state (Definition 3). Consider the generic attestation scheme in Figure 1 with the components* Read*,* Gen *and* Chk *such that*

1. Gen *is* $(N(\delta_{\mathsf{Gen}} + \delta_{\mathsf{Read}}), \varrho)$*-pseudo-random (Definition 4) and* ν_{Gen}*-unpredictable (Definition 5),*

2. Chk *is* ω*-blind second pre-image resistant (Definition 6) and* ν_{Chk}*-unpredictable (Definition 7).*

Consider an arbitrary prover $\widetilde{\mathcal{P}}$ *as in Section 3 with state* $\mathsf{State}(\widetilde{\mathcal{P}}) = \widetilde{S}$ *that can store* $\mathfrak{p}$ *memory words in its primary memory and* $\mathfrak{s}$ *memory words in its secondary memory (cf. Section 3). Let*

$$\lambda := \left|\left\{a \in \{0,1\}^{l_a} | \mathsf{Read}(\widetilde{S}, a) = \mathsf{Read}(S, a)\right\}\right| \cdot 2^{-l_a},$$

denote the fraction of state entries that are different in S *and* $\widetilde{S}$*. Then the probability of* $\widetilde{\mathcal{P}}$ *to win the security experiment* $\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}$ *(Definition 2) is upper bounded by*

$$\frac{\mathfrak{p}+\mathfrak{s}}{l_s/l_r} \cdot 2^{-(l_g+l_r)} + \max\{\omega, \nu_{\mathsf{Chk}}\} + \\ \max_{0 \leq M \leq N} \left\{(\pi(M, \mathsf{ops}) + \varrho) \cdot \gamma^{N-M} + \nu_{\mathsf{Gen}} \cdot (N - M)\right\} \quad (1)$$

where

$$\pi(n, x) := \sum_{j=\max\{0, n-2^{l_a}\}}^{n-1} \left(\max\left\{\lambda^{x+1}, \gamma\right\}\right)^{\frac{n}{x+1}-j} \cdot \binom{n}{j} \\ \cdot \left(\prod_{i=0}^{n-j} \frac{2^{l_a} - i}{2^{l_a}}\right) \cdot \left(\frac{n-j}{2^{l_a}}\right)^j \quad (2)$$

and ops *denotes the number of instructions* $\widetilde{\mathcal{P}}$ *can execute in time* $\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}}$*.*

This result implies that a software attestation scheme is ε-secure if the expression in Equation 1 is $\leq \varepsilon$, yielding a sufficient condition for security. For example if a user aims for ε-security for a prover device with fixed system parameters, he may choose the number of rounds N in dependence of an expected value of λ accordingly (cf. Appendix B).

Note that the bound given in Equation 1 gives new insights on the impact of the distribution of the state entries in S (expressed by γ) and the similarity between the expected prover state S and the actual state $\widetilde{S}$ of the prover (expressed by λ) on the security of the scheme. Both aspects have been either neglected or have been considered only informally in previous work (cf. Section 7). To provide a better intuition and to show the general applicability of Theorem 1, we compute and discuss the bound for several concrete parameters that are typical for the systems considered in the literature on software attestation in Appendix B.

Proof of Theorem 1. Let $\mathtt{Win}$ denote the event that a malicious prover $\widetilde{\mathcal{P}}$ wins the security experiment $\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}$, i.e., $\mathtt{Win}$ means that $\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}(S, l) = \textsc{accept}$. We are interested in an upper bound for $\Pr[\mathtt{Win}]$. To this end we consider several sub-cases. Let $\mathtt{Precomp}$ denote the event that the verifier $\mathcal{V}$ sends a challenge (g_0, r_0) to $\widetilde{\mathcal{P}}$ for which $\widetilde{\mathcal{P}}$ has precomputed and stored the correct response r_N in its memory (primary and/or secondary).[4] Then we have

$$\begin{aligned}\Pr[\mathtt{Win}] =& \Pr[\mathtt{Win}|\mathtt{Precomp}] \cdot \Pr[\mathtt{Precomp}] \\ &+ \Pr[\mathtt{Win}|\neg\mathtt{Precomp}] \cdot \Pr[\neg\mathtt{Precomp}] \\ \leq& \Pr[\mathtt{Precomp}] + \Pr[\mathtt{Win}|\neg\mathtt{Precomp}].\end{aligned}$$

The maximum number of responses $\widetilde{\mathcal{P}}$ can store is $\frac{\mathsf{p}+\mathsf{s}}{l_s/l_r}$. Since the challenge $(g_0, r_0) \in \{0,1\}^{l_g+l_r}$ is uniformly sampled, it follows that $\Pr[\mathtt{Precomp}] = \frac{\mathsf{p}+\mathsf{s}}{l_s/l_r} \cdot 2^{-(l_g+l_r)}$.

We now estimate the term $\Pr[\mathtt{Win}|\neg\mathtt{Precomp}]$, which we abbreviate to $\overline{\Pr}[\mathtt{Win}]$. Let $\mathtt{Correct}$ denote the event that $\widetilde{\mathcal{P}}$ determined all state entries $(s_1, \ldots, s_N)$, i.e., $s_i = \mathsf{Read}(S, a_i)$ and $(g_i, a_i) = \mathsf{Gen}(g_{i-1})$ for $i \in \{1, \ldots, N\}$ *and* that $\widetilde{\mathcal{P}}$ has executed Chk^N. Then we have

$$\overline{\Pr}[\mathtt{Win}] \leq \overline{\Pr}[\mathtt{Correct}] + \overline{\Pr}[\mathtt{Win}|\neg\mathtt{Correct}].$$

It follows from the fact that Chk^N is ω-blind second preimage resistant (Definition 6) and ν_{Chk}-unpredictable (Definition 7) that $\overline{\Pr}[\mathtt{Win}|\neg\mathtt{Correct}] \leq \max\{\omega, \nu_{\mathsf{Chk}}\}$.

For the final term $\overline{\Pr}[\mathtt{Correct}]$, we use the following claim, which we prove afterwards.

Claim 1. *The probability $\overline{\Pr}[\mathtt{Correct}]$ that $\widetilde{\mathcal{P}}$ determines all $(s_1, \ldots, s_N)$ and $r_N = \mathsf{Chk}^N(r_0, s_1, \ldots, s_N)$ in the security experiment $\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}$ under the assumption that the response to the requested challenge has not been precomputed is upper bounded by*

$$\max_{0 \leq M \leq N} \left\{ (\pi(M, \mathsf{ops}) + \varrho) \cdot \gamma^{N-M} + \nu_{\mathsf{Gen}} \cdot (N - M) \right\}$$

where $\pi(N, x)$ and ops are as in Theorem 1.

Taking these bounds together concludes the proof. □

Proof of Claim 1

We now prove Claim 1 used in the proof of Theorem 1. That is we show the claimed upper bound of $\overline{\Pr}[\mathtt{Correct}]$, which is the probability that a malicious prover $\widetilde{\mathcal{P}}$ with state $\widetilde{S} := \mathsf{State}(\widetilde{\mathcal{P}}) \neq S$ correctly determines all state entries $(s_1, \ldots, s_N)$ in the security experiment $\mathbf{Exp}^{\mathcal{A}}_{\mathsf{Attest}}$ (Definition 2) under the assumption that the response for the requested challenge has not been precomputed.

Observe that $\widetilde{\mathcal{P}}$ may decide to deviate from the protocol specification, e.g., skipping some instructions with respect to one round i (probably accepting a lower success probability for determining s_i) to save some time that could be spent on the determination of another state entry s_j with $i \neq j$ (probably aiming for a higher probability to determine s_j). Hence the challenge is to show that for *any* of these approaches the success probability does not exceed a certain (non-trivial) bound, which cannot be done by a reduction to a single assumption.

We base our proof on a sequence of games played by $\widetilde{\mathcal{P}}$ and an oracle $\mathcal{O}$ that has access to S. All these games are divided into two phases: A *setup phase* and a *challenge phase*. In the setup phase $\mathcal{O}$ generates all addresses $(a_1, \ldots, a_N)$ and determines the corresponding state entries $s_i = \mathsf{Read}(S, a_i)$. Afterwards, in the challenge phase, $\widetilde{\mathcal{P}}$ and $\mathcal{O}$ exchange several messages. In particular $\widetilde{\mathcal{P}}$ must submit its guesses $\widetilde{x}_i$ for the state entries s_i to $\mathcal{O}$. $\widetilde{\mathcal{P}}$ wins the game only if all guesses are correct, i.e., $\widetilde{x}_i = s_i$ for all $i = 1, \ldots, N$.

The differences between the games lie in the possibilities of $\widetilde{\mathcal{P}}$ to deviate from the protocol specification. While these possibilities are quite limited in the first game (Game 0), $\widetilde{\mathcal{P}}$ gets more and more control with each subsequent game and thus can to perform more powerful attacks. For each transformation between two consecutive games, we show how the success probability of $\widetilde{\mathcal{P}}$ changes. In most cases it turns out that the previous game represents a subset of the possible attack strategies of the current game. Note that $\mathcal{O}$ formally represents the honest execution of certain parts of the protocol and should not be confused with a real party. Consequently, we assume that transferring messages between $\widetilde{\mathcal{P}}$ and $\mathcal{O}$ takes no time.

Observe that the intention of $\mathcal{O}$ is to have an elegant method for ignoring all computations of $\widetilde{\mathcal{P}}$ which are honestly executed by assumption. Hence to exclude artificial attacks where $\widetilde{\mathcal{P}}$ uses the time and/or memory gained by outsourcing the computation to $\mathcal{O}$, we restrict the time-bound and the size of the primary memory of $\widetilde{\mathcal{P}}$ to what is necessary for honestly executing those computations that are outsourced to $\mathcal{O}$.

Game 0: Randomly Sampling Addresses in Regular Time Intervals.

Game Description. The purpose of this game is to investigate provers $\widetilde{\mathcal{P}}$ which (1) do not exploit any aspects related to the execution of Gen and (2) that are forced to use exactly time δ_{Read} for the determination of each state entry s_i. This is captured by modelling the game as follows: Within the setup phase, $\mathcal{O}$ samples pairwise independent and uniform addresses $(a_1, \ldots, a_N)$ and sets $s_i := \mathsf{Read}(S, a_i)$ for all $i \in \{1, \ldots, N\}$. In the challenge phase, $\mathcal{O}$ iteratively queries $\widetilde{\mathcal{P}}$ with a_i and $\widetilde{\mathcal{P}}$ returns some response $\widetilde{x}_i$.

Hereby, $\widetilde{\mathcal{P}}$ can access the Read oracle, which on input a returns $s = \mathsf{Read}(\widetilde{S}, a)$ after time δ_{Read}. Since this is the only operation expected from an honest prover, the size of the primary memory only allows to store an address a and a state entry s. Moreover the total time-bound is limited to $N \cdot \delta_{\mathsf{Read}}$, meaning that $\widetilde{\mathcal{P}}$ automatically fails if it needs more time in total than this bound.

Observe that $\mathcal{O}$ ensures that $\widetilde{\mathcal{P}}$ cannot change the order of the memory addresses, i.e., $\mathcal{O}$ only sends a_i to $\widetilde{\mathcal{P}}$ after a_{i-1}

[4] More precisely, $\mathcal{A}$ has precomputed this value during the preparation phase and stored the response as part of $\widetilde{S}$.

has been sent.[5] We denote with *round* i the time-frame between the point in time where $\widetilde{\mathcal{P}}$ receives a_i and the point in time where $\widetilde{\mathcal{P}}$ receives a_{i+1} for $i \in \{1, \ldots, N-1\}$. With *round* N we denote the time-frame between the point in time where $\widetilde{\mathcal{P}}$ receives a_N and the point in time where $\widetilde{\mathcal{P}}$ sends the last response $\widetilde{x}_N$ to $\mathcal{O}$. $\widetilde{\mathcal{P}}$ wins the game if (1) $\widetilde{x}_i = s_i$ for all $i \in \{1, \ldots, N\}$ and (2) each round took at most time δ_{Read}. Otherwise $\widetilde{\mathcal{P}}$ looses the game.

Success Probability. We are interested in an upper bound for the probability $\Pr[\mathtt{Win}_0]$ that $\widetilde{\mathcal{P}}$ wins Game 0. Since $\widetilde{\mathcal{P}}$ looses for sure when he uses more time than δ_{Read} to respond to a_i in at least one round i, it is sufficient to restrict to provers that take at most time δ_{Read} in each round. To this end, we derive an upper bound which allows to treat the individual rounds separately. We start with the final round N and distinguish between two cases.

In Case 1 the response $\widetilde{x}_N$ is the direct result of a query to the Read oracle, i.e., $\widetilde{x}_N = \mathsf{Read}(\widetilde{S}, a)$ for some address a. If $a = a_N$ the probability of $\widetilde{x}_N := \mathsf{Read}(\widetilde{S}, a_N) = s_N := \mathsf{Read}(S, a_N)$ is λ (cf. Theorem 1) since a_N is sampled uniformly and independently from the previous addresses. Now consider that $a \neq a_N$. Since $\widetilde{x}_N = \mathsf{Read}(\widetilde{S}, a)$ and due to the fact that $\widetilde{\mathcal{P}}$ must respond with $\widetilde{x}_N$ in time δ_{Read} after receiving a_N, $\widetilde{\mathcal{P}}$ has no time left to perform any other instructions than Read during round N. In particular a could not be chosen in dependence of a_N, hence being independent of a_N. Then $\widetilde{x}_N = s_N$ happens with probability of at most γ (cf. Definition 3). It follows that in Case 1 the probability $\Pr[\mathtt{Win}_0]$ is upper bounded by $\max\{\lambda, \gamma\} \cdot \Pr[\widetilde{x}_1 = \mathsf{Read}(S, a_1) \wedge \ldots \wedge \widetilde{x}_{N-1} = \mathsf{Read}(S, a_{N-1})]$.

Next we consider Case 2, where $\widetilde{x}_N$ is not the result of a query to the Read oracle. It follows from the incompressibility of S (Definition 3) and the fact that a_N has been sampled uniformly and independent of all previous addresses a_i with $i < N$, that the probability of $\widetilde{x}_N = \mathsf{Read}(S, a_N)$ is upper bounded by γ. Hence,

$$\gamma \cdot \Pr[\widetilde{x}_1 = \mathsf{Read}(S, a_1) \wedge \ldots \wedge \widetilde{x}_{N-1} = \mathsf{Read}(S, a_{N-1})]$$

is an upper bound of $\Pr[\mathtt{Win}_0]$ in Case 2. It follows from Cases 1 and 2 that

$$\Pr[\mathtt{Win}_0] \leq \max\{\lambda, \gamma\} \cdot \Pr\left[\widetilde{x}_1 = \mathsf{Read}(S, a_1) \right. \\ \left. \wedge \ldots \wedge \widetilde{x}_{N-1} = \mathsf{Read}(S, a_{N-1})\right]$$

and by induction $\Pr[\mathtt{Win}_0] \leq \pi_0 = \pi_0(N) := (\max\{\lambda, \gamma\})^N$.

Game 1: Prover Controls the Address Generation Time.

Game Description. In this game we increase the power of the malicious prover $\widetilde{\mathcal{P}}$ and allow him to freely choose how much time he devotes to determine each value s_i, as long as the total time for determining $(s_1, \ldots, s_N)$ does not exceed $N \cdot \delta_{\mathsf{Read}}$. This reflects the fact that in the attestation protocol, a malicious prover $\widetilde{\mathcal{P}}$ may generate the memory addresses $(a_1, \ldots, a_N)$ on its own whenever it wants to.

Formally, this is captured by introducing a *req* protocol message which $\widetilde{\mathcal{P}}$ needs to send to $\mathcal{O}$ for receiving the next address a_i during the challenge phase. More precisely, $\mathcal{O}$ sends a_i to $\widetilde{\mathcal{P}}$ only when $\widetilde{\mathcal{P}}$ sent the i-th request *req* to $\mathcal{O}$.

Since each round may take a different time period, the winning conditions are relaxed by replacing the time restriction on the individual rounds by an overall time-bound for the entire challenge phase. This means that $\widetilde{\mathcal{P}}$ wins Game 1 if (1) $\widetilde{x}_i = s_i$ for all $i \in \{1, \ldots, N\}$ and (2) the duration of the challenge phase does not exceed the time $N \cdot \delta_{\mathsf{Read}}$. The size of the primary memory remains as in Game 0.

Success Probability. We now upper bound the probability $\Pr[\mathtt{Win}_1]$ that $\widetilde{\mathcal{P}}$ wins Game 1. To this end, we divide the number of rounds into four distinct sets. Let N_{coll} denote the number of rounds where the address sampled by $\mathcal{O}$ is equal to an address of some previous round by coincidence, i.e., $N_{\text{coll}} := |\{i \in \{2, \ldots, N\} | \exists j \in \{1, \ldots, i-1\} : a_i = a_j\}|$.

With respect to the remaining $N - N_{\text{coll}}$ rounds, let N_{equal} (resp. N_{more}, resp. N_{less}) be the number of rounds where $\widetilde{\mathcal{P}}$ responds in time equal (resp. more, resp. less) than δ_{Read}. Thus we have $N = N_{\text{coll}} + N_{\text{equal}} + N_{\text{less}} + N_{\text{more}}$.

Let $\mathtt{Coll}(N_{\text{coll}})$ denote the event that exactly N_{coll} of the N addresses are equal to some previous addresses. This implies that in $N - N_{\text{coll}}$ rounds pairwise different addresses are sampled. Moreover, since there are only 2^{l_a} different addresses, $N - N_{\text{coll}}$ is upper bound by 2^{l_a}. It follows that $N - N_{\text{coll}} \leq 2^{l_a} \Leftrightarrow N_{\text{coll}} \geq N - 2^{l_a}$. Thus it must hold that $N_{\text{coll}} \geq \max\{0, N - 2^{l_a}\}$. Hence, $\Pr[\mathtt{Win}_1]$ is equal to

$$\sum_{N_{\text{coll}} = \max\{0, N - 2^{l_a}\}}^{N-1} \Pr[\mathtt{Win}_1 | \mathtt{Coll}(N_{\text{coll}})] \cdot \Pr[\mathtt{Coll}(N_{\text{coll}})].$$

We now derive upper bounds for $\Pr[\mathtt{Win}_1 | \mathtt{Coll}(N_{\text{coll}})]$ and $\Pr[\mathtt{Coll}(N_{\text{coll}})]$.

In general, $\Pr[\mathtt{Coll}(N_{\text{coll}})]$ can be expressed by (number combinations of rounds with equal addresses) × (probability that addresses in $N - N_{\text{coll}}$ rounds are pairwise different) × (probability that addresses in the remaining rounds are equal to some previous address). The first term is at most $\binom{N}{N_{\text{coll}}}$ while an upper bound for the last term is $\left(\frac{N - N_{\text{coll}}}{2^{l_a}}\right)^{N_{\text{coll}}}$. This gives an upper bound

$$\binom{N}{N_{\text{coll}}} \cdot \left(\prod_{i=0}^{N - N_{\text{coll}}} \frac{2^{l_a} - i}{2^{l_a}}\right) \cdot \left(\frac{N - N_{\text{coll}}}{2^{l_a}}\right)^{N_{\text{coll}}} . \quad (3)$$

for $\Pr[\mathtt{Coll}(N_{\text{coll}})]$ if $\max\{0, N - 2^{l_a}\} \leq N_{\text{coll}} \leq N - 1$. We now fix a value for N_{coll} and aim for an upper bound for the probability $\Pr[\mathtt{Win}_1 | \mathtt{Coll}(N_{\text{coll}})]$. We do so by giving separate upper bounds on the success probability for the four different types of rounds. Let $\mathtt{ops} = \mathtt{ops}(\delta_{\mathsf{Read}})$ be the number of instructions that can be executed by the computing engine of $\widetilde{\mathcal{P}}$ in time δ_{Read}. Since we are interested in an *upper bound* of $\widetilde{\mathcal{P}}$'s success probability, we make several assumptions in favor of $\widetilde{\mathcal{P}}$: (1) For rounds where $\widetilde{\mathcal{P}}$ invested more time than δ_{Read}, we use the trivial upper bound of 1 even if the time period exceeded δ_{Read} only by the time required to execute one single instruction. (2) For rounds where the requested address coincides with an address previously asked, we likewise use the bound of 1. Moreover we assume that these rounds take *no* time at all and the $\mathtt{ops}$ instructions saved can be used in $\mathtt{ops}$ other rounds. (3) In rounds that take less time than δ_{Read}, it follows from the incompressibility of S (Definition 3) and the fact that all addresses are pairwise distinct that $\widetilde{x}_i = s_i$ with probability $\leq \gamma$. Again, we assume that these rounds take *no* time at all and that

[5] This is a consequence of Remark 7.

the $\mathtt{ops}$ instructions saved can be used in $\mathtt{ops}$ other rounds. (4) In a round that takes exactly time δ_{Read} $\widetilde{\mathcal{P}}$ succeeds at most with probability $\max\{\lambda, \gamma\}$ (cf. Game 0).

While these assumptions strongly exaggerate the possibilities of $\widetilde{\mathcal{P}}$, they allow to identify optimum strategies. More precisely for each round where $\widetilde{\mathcal{P}}$ uses less time than δ_{Read} or where a previously asked address is requested again, the best approach is to spend the $\mathtt{ops}$ saved instructions in $\mathtt{ops}$ other rounds such that for each of these rounds the probability of correctly determining s_i is equal to 1. It follows that $N_{\text{more}} = \mathtt{ops} \cdot (N_{\text{coll}} + N_{\text{less}})$ and hence $N = N_{\text{coll}} + N_{\text{equal}} + N_{\text{less}} + N_{\text{more}} = N_{\text{equal}} + (\mathtt{ops} + 1) \cdot (N_{\text{coll}} + N_{\text{less}})$. Hence, we have

$$\Pr\left[\mathtt{Win}_1 | \mathtt{Coll}(N_{\text{coll}})\right] \leq \pi_0(N_{\text{equal}}) \cdot \gamma^{N_{\text{less}}} \cdot 1^{N_{\text{coll}}+N_{\text{more}}} =$$
$$\max_{N_{\text{less}}} \left\{ \lambda^{N-(\mathtt{ops}+1)\cdot(N_{\text{coll}}+N_{\text{less}})} \cdot \gamma^{N_{\text{less}}}, \gamma^{N-(\mathtt{ops}+1)\cdot N_{\text{coll}} - \mathtt{ops}\cdot N_{\text{less}}} \right\}$$
$$\overset{\text{cf. Apx. A}}{=} \left(\max\left\{ \lambda^{\mathtt{ops}(\delta_{\mathsf{Read}})+1}, \gamma \right\} \right)^{\frac{N}{\mathtt{ops}(\delta_{\mathsf{Read}})+1} - N_{\text{coll}}}, \quad (4)$$

where the last equation is shown in Appendix A. This shows that $\pi\left(N, \mathtt{ops}(\delta_{\mathsf{Read}})\right)$ is an upper bound for the probability $\Pr\left[\mathtt{Win}_1\right]$, where $\pi(n, x)$ is defined as in Equation 2. Observe that for any fixed value of N_{coll}, the probability of having N_{coll} collisions (Equation 3) increases with N (as long as $N_{\text{coll}} \geq \max\{0, N - 2^{l_a}\}$) while the probability to determine the values $(s_1, \ldots, s_N)$ (Equation 4) decreases for N.

Game 2: Skipping Address Generation.

Game Description. So far we covered only provers $\widetilde{\mathcal{P}}$ that honestly generate all addresses $(a_1, \ldots, a_N)$. Now we change the game such that $\widetilde{\mathcal{P}}$ may decide in each round i to skip the generation of address a_i. This allows $\widetilde{\mathcal{P}}$ to "buy" more time for determining the values s_i but at the "cost" of not knowing a_i. Formally this is captured by defining a second message *skip* besides *req*. Specifically, in each round i of the challenge phase, $\widetilde{\mathcal{P}}$ either sends *req* or *skip*. In case of *req*, $\mathcal{O}$ behaves as in Game 1 and sends the next a_i to $\widetilde{\mathcal{P}}$. However, when $\widetilde{\mathcal{P}}$ sends *skip* then $\mathcal{O}$ does *not* send a_i to $\widetilde{\mathcal{P}}$ and extends the time-bound by δ_{Gen}. That is, at the beginning of the challenge phase, the winning conditions are that (1) all responses $(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ of $\widetilde{\mathcal{P}}$ are correct, i.e., $\widetilde{x}_i = s_i \; \forall i \in \{1, \ldots, N\}$ and (2) the challenge phase does not take more time than $N \cdot \delta_{\mathsf{Read}}$. However each time $\widetilde{\mathcal{P}}$ sends a *skip* message to $\mathcal{O}$, the time-bound is extended by δ_{Gen}.

Success Probability. We now determine the probability $\Pr\left[\mathtt{Win}_2\right]$ that $\widetilde{\mathcal{P}}$ wins Game 2. To this end we follow the same line of arguments as in Game 1. The only difference is that rounds where collisions in the addresses took place or where either Read or Gen have been skipped take no time at all and free $\mathtt{ops}(\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}})$ instructions for other rounds. That is we get a bound with the same structure as in Game 1 but where $\mathtt{ops}(\delta_{\mathsf{Read}})$ is replaced by $\mathtt{ops}(\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}})$, i.e., $\Pr\left[\mathtt{Win}_2\right] \leq \pi\left(N, \mathtt{ops}(\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}})\right)$.

Game 3: Replacing the Random Sampling with Gen .

Game Description. Now we consider a variant of Game 2 with the only difference being that the addresses $(a_1, \ldots, a_N)$ are generated by Gen instead of being randomly sampled by $\mathcal{O}$. That is, during the setup phase $\mathcal{O}$ randomly samples g_0 and generates $(a_1, \ldots, a_N)$ using Gen.

Success Probability. Let $\Pr\left[\mathtt{Win}_3\right]$ be the probability that $\widetilde{\mathcal{P}}$ wins Game 3. Using a standard argument, it follows from the pseudo-randomness of the outputs of Gen (Definition 4) that $\left|\Pr\left[\mathtt{Win}_3\right] - \Pr\left[\mathtt{Win}_2\right]\right| \leq \varrho$ and hence $\Pr\left[\mathtt{Win}_3\right] \leq \Pr\left[\mathtt{Win}_2\right] + \varrho \leq \pi\left(N, \mathtt{ops}(\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}})\right) + \varrho$.

Game 4: Giving Access to Gen.

Game Description. In the final game $\mathcal{O}$ no longer generates $(a_1, \ldots, a_N)$ for $\widetilde{\mathcal{P}}$. Instead $\widetilde{\mathcal{P}}$ now queries the Gen oracle, which on input g_i returns $(g_i, a_i) = \mathsf{Gen}(g_{i-1})$ after time δ_{Gen}. To this end, $\mathcal{O}$ samples g_0 in the setup phase and gives this value to $\widetilde{\mathcal{P}}$.

Observe that the size of the primary memory of $\widetilde{\mathcal{P}}$ is increased to additionally store a value g. Further, the time-bound of the challenge phase is increased to $N \cdot (\delta_{\mathsf{Gen}} + \delta_{\mathsf{Read}})$.

Success Probability. The only difference between Game 4 and Game 3 is that $\widetilde{\mathcal{P}}$ now knows g_0 and can query the Gen oracle. Recall that g_0 is used by Gen for computing $(a_1, \ldots, a_N)$. Hence $\widetilde{\mathcal{P}}$ may decide to skip the generation of one or more addresses and save the time and memory for other computations. However, since Gen is assumed to be ν_{Gen}-unpredictable (Definition 5), $\widetilde{\mathcal{P}}$ cannot derive any information on a_{i+1} or g_{i+1} from g_i without querying Gen. Thus if $\widetilde{\mathcal{P}}$ never queries Gen with some value g_i it cannot distinguish the subsequent values $(g_{i+1}, \ldots, g_N)$ with a probability better than $(N - i) \cdot \nu_{\mathsf{Gen}}$. Therefore we can restrict to provers that compute $(a_1, g_1), \ldots, (a_M, g_M)$ and that skip $(a_{M+1}, g_{M+1}), \ldots, (a_N, g_N)$.

Let $\Pr\left[\mathtt{Win}_4\right]$ be the probability to win Game 4 and let $\Pr\left[\mathtt{Win}_4(M)\right]$ be the probability to win Game 4 for a fixed M. That is we have $\Pr\left[\mathtt{Win}_4\right] \leq \max_M \left\{\Pr\left[\mathtt{Win}_4(M)\right]\right\}$. Now consider a variation of Game 4 where $\mathcal{O}$ replaces the values $(a_{M+1}, g_{M+1}), \ldots, (a_N, g_N)$ by independent and uniformly sampled values and we denote with $\Pr\left[\mathtt{Win}_4'(M)\right]$ the probability that $\widetilde{\mathcal{P}}$ wins this game. Since Gen is assumed to be ν_{Gen}-unpredictable (cf. Definition 5), it holds that $\Pr\left[\mathtt{Win}_4(M)\right] \leq \Pr\left[\mathtt{Win}_4'(M)\right] + \nu_{\mathsf{Gen}} \cdot (N - M)$.

With respect to $\Pr\left[\mathtt{Win}_4(M)\right]$, observe that for the first M rounds the situation is as in Game 3. Hence the success probability for the first M rounds is upper bounded by $\pi\left(M, \mathtt{ops}(\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}})\right) + \varrho$. For the remaining $N - M$ rounds, $\mathcal{O}$ uses uniformly sampled values $(a_{M+1}, \ldots, a_N)$ that are unknown to $\widetilde{\mathcal{P}}$. Hence the probability of $\widetilde{\mathcal{P}}$ deriving $(s_{M+1}, \ldots, s_N)$ correctly is upper bounded by γ^{N-M}. This yields $\Pr\left[\mathtt{Win}_4'(M)\right] \leq \left(\pi\left(M, \mathtt{ops}(\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}})\right) + \varrho\right) \cdot \gamma^{N-M}$ and hence $\Pr\left[\mathtt{Correct}\right]$ is at most

$$\max_{0 \leq M \leq N} \left\{ \left(\pi(M, \mathtt{ops}(\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}})) + \varrho\right) \cdot \gamma^{N-M} + \nu_{\mathsf{Gen}} \cdot (N - M) \right\}.$$

7. DISCUSSION AND CONCLUSION

We presented the first formal security framework for software attestation and precisely defined various of the underlying system and design parameters. Moreover we presented a generic software attestation protocol that encompasses most existing schemes in the literature. For this generic scheme we derived an upper bound on the success probability of a malicious prover that depends on the formalized parameters. We see the generic scheme and its proof as blueprints that can be adapted to concrete instantiations.

One lesson learned is the impact of these parameters on the security of the generic scheme. For example, it has been observed before that free memory should be filled with pseudo-random data [24] and a later code-compression attack [3] indicated that code redundancy also impacts security. However, the attack was dismissed as impractical [16] and ignored in later works [17, 29]. In contrast, we consider the general probability distribution of the state (code and data) in Definition 3 and directly connect it to the adversary advantage. As a result, one can directly evaluate the honest prover state and determine whether additional measures to achieve state incompressibility, such as filling free memory with pseudo-random data, are required to prevent memory copy and code compression attacks.

Our results also show that traditional cryptographic assumptions are partially too strong (second pre-image resistance) and partially too weak (pseudo-randomness). Further, we identified new (sufficient) conditions on the core functionalities of software attestation. Most previous works require the software attestation algorithm to iterate over all memory words of the secondary memory without giving any formal justification. Our bound allows to identify lower values for N (if the other parameters are known), enabling more efficient solutions that provide a tradeoff between the number of iterations N and the success probability of a malicious prover. Thus our work represents the first step towards efficient and provably secure software attestation schemes.

Still, several open questions remain for future work. One being to relax the presented conditions or to derive necessary conditions. A further task is to determine concrete instantiations. While Gen and Chk could be easily realized on devices that provide hardware-assisted cryptographic functions, such as block cipher implementations in hardware similar to the AES instructions in modern CPUs [30], this becomes more challenging on other platforms.

We are currently working on the following aspects: (1) a practical instantiation of the generic software attestation scheme and its evaluation and (2) the evaluation of existing software attestation schemes in our framework.

8. REFERENCES

[1] T. AbuHmed, N. Nyamaa, and D. Nyang. Software-based remote code attestation in wireless sensor network. In *Global Telecommunications (GLOBECOM)*. IEEE, 2009.

[2] Atmel. tinyAVR homepage. http://www.atmel.com/tinyavr/, 2013.

[3] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In *Computer and Communications Security (CCS)*. ACM, 2009.

[4] A. Datta, J. Franklin, D. Garg, and D. Kaynar. A logic of secure systems and its application to trusted computing. In *Security and Privacy, 2009 30th IEEE Symposium on*, 2009.

[5] K. E. Defrawy, A. Francillon, D. Perito, and G. Tsudik. SMART: Secure and minimal architecture for (establishing a dynamic) root of trust. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2012.

[6] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology – CRYPTO*. Springer, 1993.

[7] C. Dwork, M. Naor, and H. Wee. Pebbling and proofs of work. In *Advances in Cryptology – CRYPTO*. Springer, 2005.

[8] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. Systematic treatment of remote attestation. Cryptology ePrint Archive, Report 2012/713, 2012. http://eprint.iacr.org/.

[9] J. Franklin, M. Luk, A. Seshadri, and A. Perrig. PRISM: Human-verifieable code execution. Technical report, Carnegie Mellon University, 2007.

[10] 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.

[11] J. T. Giffin, M. Christodorescu, and L. Kruger. Strengthening software self-checksumming via self-modifying code. In *Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2005.

[12] V. Graizer and D. Naccache. Alien vs. Quine. *Security Privacy, IEEE*, 5(2):26 –31, 2007.

[13] V. Gratzer and D. Naccache. Alien vs. Quine, the vanishing circuit and other tales from the industry's crypt. In *Advances in Cryptology – EUROCRYPT*. Springer, 2007. Invited Talk.

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

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

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

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

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

[27] R. Strackx, F. Piessens, and B. Preneel. Efficient isolation of trusted subsystems in embedded systems. In *Security and Privacy in Communication Networks (SecureComm)*. Springer, 2010.

[28] Trusted Computing Group (TCG). *TPM Main Specification, Version 1.2*, 2011.

[29] A. Vasudevan, J. Mccune, J. Newsome, A. Perrig, and L. V. Doorn. CARMA: A hardware tamper-resistant isolated execution environment on commodity x86 platforms. In *ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*. ACM, 2012.

[30] Wikipedia. AES instruction set. `http://en.wikipedia.org/wiki/AES_instruction_set`, 2013.

[31] Q. Yan, J. Han, Y. Li, R. H. Deng, and T. Li. A software-based root-of-trust primitive on multicore platforms. In *ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*. ACM, 2011.

[32] 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.

APPENDIX

A. DETAILS OF THE PROOF

We prove the upper bound of $\Pr\left[\mathtt{Win}_1 | \mathtt{Coll}(N_{\text{coll}})\right]$ used in Game 1 of the proof of Theorem 1. Observe that $0 \leq N_{\text{less}}$ and $0 \leq N_{\text{equal}} = N - (\mathsf{ops}+1) \cdot (N_{\text{coll}} + N_{\text{less}}) \Leftrightarrow N_{\text{less}} \leq \frac{N}{\mathsf{ops}+1} - N_{\text{coll}}$, i.e., $0 \leq N_{\text{less}} \leq \frac{N}{\mathsf{ops}+1} - N_{\text{coll}}$. To simplify the first term $\lambda^{N-(\mathsf{ops}+1)\cdot(N_{\text{coll}}+N_{\text{less}})} \cdot \gamma^{N_{\text{less}}}$, we define $e := \log_\lambda(\gamma)$ and rephrase it as $\lambda^{N-(\mathsf{ops}+1)\cdot N_{\text{coll}}-(\mathsf{ops}+1-e)\cdot N_{\text{less}}}$.

When $\mathsf{ops}+1-e < 0$, the maximum is achieved for $N_{\text{less}} = 0$, hence in this case the upper bound is $\lambda^{N-(\mathsf{ops}+1)\cdot N_{\text{coll}}}$. In the other case we get an upper bound for $N_{\text{less}} = \frac{N}{\mathsf{ops}+1} - N_{\text{coll}}$, yielding

$$\lambda^{N-(\mathsf{ops}+1)\cdot N_{\text{coll}}-(\mathsf{ops}+1-e)\cdot\left(\frac{N}{\mathsf{ops}+1}-N_{\text{coll}}\right)} = \gamma^{\frac{N}{\mathsf{ops}+1}-N_{\text{coll}}}.$$

With respect to the second term, i.e., $\gamma^{N-(\mathsf{ops}+1)\cdot N_{\text{coll}}-\mathsf{ops}\cdot N_{\text{less}}}$, the maximum value is achieved if N_{less} is as big as possible, i.e., $N_{\text{less}} = \frac{N}{\mathsf{ops}+1} - N_{\text{coll}}$. This gives an upper bound of

$$\gamma^{N-(\mathsf{ops}+1)\cdot N_{\text{coll}}-\mathsf{ops}\cdot N_{\text{less}}} = \gamma^{\frac{N}{\mathsf{ops}+1}-N_{\text{coll}}}.$$

Altogether, it follows that

$$\Pr\left[\mathtt{Win}_1 | \mathtt{Coll}(N_{\text{coll}})\right] \leq \left(\max\left\{\lambda^{\mathsf{ops}(\delta_{\mathsf{Read}})+1}, \gamma\right\}\right)^{\frac{N}{\mathsf{ops}(\delta_{\mathsf{Read}})+1}-N_{\text{coll}}}.$$

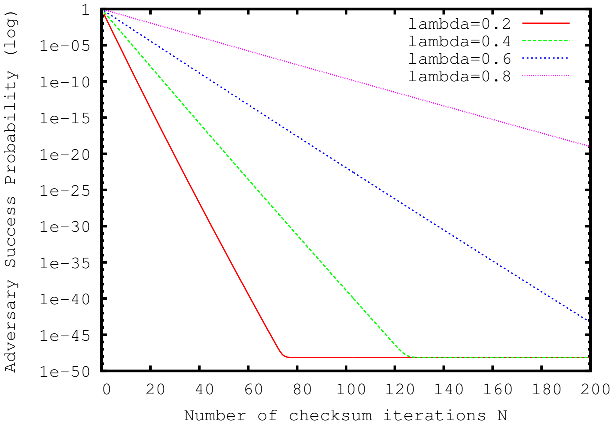


Figure 2: Upper Bound (Equation 1) for different values of N and λ.

B. EXAMPLE FOR UPPER BOUND

To get a better intuition of the upper bound (Equation 1) given in Theorem 1, especially on the impact of the similarity of the malicious prover's state to the honest prover's state expressed by λ and the number of rounds N, we provide some concrete examples in this section.

At first we have to fix various parameters. To this end, we consider typical parameters for l_g and l_r that can be found in the literature on software attestation. Moreover, we assume that all cryptographic primitives are *perfectly secure* and that the values in S are uniformly distributed:

$$\omega := 2^{-l_r},\ \nu_{\mathsf{Chk}} = 0,\ \varrho := 0,\ \gamma := 2^{-l_s},\ \nu_{\mathsf{Gen}} := 0.$$

The bound in Equation 1 then simplifies to

$$\frac{\mathfrak{p}+\mathfrak{s}}{l_s/l_r \cdot 2^{(l_g+l_r)}} + 2^{-l_r} + \max_{0\leq M\leq N}\left\{\left(\pi\left(M, \mathsf{ops}\right)\right) \cdot 2^{-l_s\cdot(N-M)}\right\}.$$

Recall that the value ops has been defined as the number of operations a prover can perform in time $\delta_{\mathsf{Read}} + \delta_{\mathsf{Gen}}$. It was used in the proof to tackle the following question: *If an attacker decides to skip one round, for how many other rounds can he increase his probability of success?* While ops certainly expresses an upper bound on this number (the adversary has to spend at least one instruction per round), it certainly is an overestimation of the adversary's capabilities. Hence, we set $\mathsf{ops} = 2$ to get more meaningful results. This represents an adversary who can win two rounds if he skips another round.

Recall that λ expresses the fraction of state entries where the state of the malicious prover matches with the state S of the honest prover. We exemplarily use $\lambda \in \{0.2, \ldots, 0.8\}$. As shown in Figure 2, for small values of λ (i.e., for malicious provers with a state that is quite different from the honest prover's state), a relatively low number of rounds is sufficient to achieve a reasonably low adversary advantage. However, for large values of λ more rounds are required. Further, for the chosen system parameters, the advantage seems to converge to a minimal adversary advantage of 10^{-48}.

Observe that in the literature, it is often suggested to use $N = \log(\mathfrak{s}) \cdot \mathfrak{s}$ rounds. Interestingly, our experiments indicate that significantly less rounds can be sufficient.