

Probabilistic and Considerate Attestation of IoT Devices against Roving Malware

Xavier Carpent
UC Irvine
xcarpent@uci.edu

Norrathep Rattanavipanon
UC Irvine
nrattana@uci.edu

Gene Tsudik
UC Irvine
gene.tsudik@uci.edu

Abstract—Remote Attestation (RA) is a popular means of detecting malware presence (or verifying its absence) on embedded and IoT devices. It is especially relevant to low-end devices that are incapable of protecting themselves against infection. Malware that is aware of ongoing or impending attestation and aims to avoid detection can relocate itself during computation of the attestation measurement. In order to thwart such behavior, prior RA techniques are either non-interruptible or explicitly forbid modification of storage during measurement computation. However, since the latter can be a time-consuming task, this curtails availability of device’s other (main) functions, which is especially undesirable, or even dangerous, for devices with time- and/or safety-critical missions.

In this paper, we propose *SMARM*, a light-weight technique, based on shuffled measurements, as a defense against roving malware. In *SMARM*, memory is measured in a randomized and secret order. This does not impact device’s availability – the measurement process can be interrupted, even by malware, which can relocate itself at will. We analyze various malware behaviors and show that, while malware can escape detection in a single attestation instance, it is highly unlikely to avoid eventual detection.

1. Introduction

In the past, malware primarily targeted general-purpose computers and smartphones. In recent years, the number and variety of various specialized computing devices has increased dramatically. This includes all kinds of embedded devices, cyber-physical systems (CPS) and Internet-of-Things (IoT) gadgets. They can be encountered in many diverse “smart” settings, such as home, office, factory, automotive and public venues. Unfortunately, these devices represent natural and attractive targets for malware, mainly because they are often numerous, connected to the Internet and/or inter-connected, and their security is poor or non-existent.

As society becomes increasingly accustomed to being surrounded by, and deriving benefits from, such devices, their well-being becomes a paramount concern. In the context of actuation-capable devices, malware can impact security and safety, e.g., as demonstrated by Stuxnet [16]. Whereas, for sensing devices, malware can undermine pri-

vacy by obtaining ambient information. Also, malware can turn vulnerable IoT devices into zombies that can become sources for DDoS attacks.¹

Security is typically not a priority for low-end device manufacturers, due to cost, size or power constraints, as well as the rush-to-market syndrome. It is thus unrealistic to expect low-end IoT devices to have the means to prevent malware attacks. The next best thing is detection of malware presence, which typically requires some form of Remote Attestation (RA). RA is a security service that involves a trusted entity (called \mathcal{Vrf}) that securely verifies the status of a remote device (called \mathcal{Prv}) and detects malware presence on the latter. If malware is detected, \mathcal{Prv} ’s software can be re-set or rolled back and measures can be taken to prevent similar infections.

1.1. Remote Attestation (RA)

RA techniques generally operate as follows: \mathcal{Vrf} sends an attestation request (that contains a challenge) to \mathcal{Prv} . Once it receives the request, \mathcal{Prv} invokes some trusted attestation code (*AttC*) that computes a measurement over \mathcal{Vrf} -specified memory region M . This measurement (often called an attestation token) is typically realized as a Message Authentication Code (MAC) over M computed using a secret key K , which is protected from unauthorized access by the security architecture of \mathcal{Prv} . (The goal of \mathcal{Prv} ’s underlying security architecture is to prevent malware \mathcal{Mal} from forging measurements.) The measurement is then returned to \mathcal{Vrf} which, in turn, determines whether \mathcal{Vrf} is in a healthy or compromised state.

RA typically comes in three flavors: software-based, hardware-based and hybrid. Since this work is related to hybrid attestation, we refer to [4] for a discussion of the other two types. Hybrid attestation refers to RA techniques that aim to minimize hardware requirements on \mathcal{Prv} , in part to be suitable for low-end devices. Notable hybrid attestation architectures are: SMART [7], TrustLite [12], and TyTAN [2].

SMART is the first hybrid RA architecture. It stipulates that attestation code *AttC* and attestation key K are stored in ROM and guarded by hard-wired MCU access control

1. About a year ago, hackers used a multitude of compromised “smart” cameras and DVRs to mount a massive-scale DDoS attack.

rules. These rules enforce that only AttC has access to K , and that AttC is atomic, i.e., non-interruptible, and executed as a whole.

TrustLite differs from SMART in that interrupts are allowed and handled securely by the CPU Exception Engine. Also, access control rules can be programmed using an Execution-Aware Memory Protection Unit (EA-MPU). TyTAN further builds on TrustLite, notably featuring dynamic configuration of access control rules.

1.2. Motivation

Since low-end devices are often used in real-time and safety-critical applications, it is important to minimize the impact of security on normal operation (i.e., availability) of such devices. In particular, it might be undesirable to allow AttC on $\mathcal{P}rv$ to run without interruption, considering that computing a measurement over a substantial amount of memory might take a relatively long time. In other words, AttC should be interruptible by a legitimate, time-critical application.

However, $\mathcal{P}rv$ might have been compromised and the same time-critical application might contain malware ($\mathcal{M}al$). $\mathcal{M}al$ presumably wants to evade detection. When confronted with attestation, it may want to simply erase itself, perhaps in order to reappear later. Alternatively, it might remain on $\mathcal{P}rv$ and try to avoid detection by relocating itself *during* attestation. This behavior corresponds to *roving malware*.

To this end, in this paper, we focus on reconciling two seemingly contradicting objectives: resistance against roving malware and minimizing the impact of attestation on $\mathcal{P}rv$'s availability.

Prior hybrid RA designs had different motivations. SMART avoids roving malware by enforcing non-interruptibility of the measuring process, which fully sacrifices interruptibility by a time-critical task. Although TrustLite allows secure interrupts, it fails to detect roving malware. Whereas, TyTAN protects against roving malware by enforcing a rule that the process being measured can not interrupt AttC , while other processes can. Nonetheless, this approach lacks transparency of attestation, i.e., if a time-critical process is being measured, it can not interrupt the attestation process. Also, TyTAN might not detect roving malware if process isolation is compromised (e.g., by a kernel bug) resulting in process collusion. Furthermore, TrustLite and (to a lesser degree) TyTAN require more advanced hardware features than SMART, which translates into extra cost for low-end platforms.

2. *SMART*: RA via Shuffled Measurements

To mitigate the conflict between roving malware ($\mathcal{M}al$) detection and critical mission of $\mathcal{P}rv$, we adopt an approach whereby the attestation process is **interruptible**, while the order in which M is measured is determined randomly and privately, by the attestation process. The rationale is that, if $\mathcal{M}al$ remains unaware of what portions of M have been already measured (covered), it can not decide where

to relocate itself to escape detection. $\mathcal{M}al$'s optimal strategy (i.e., where and when to relocate) depends on its knowledge of the attestation coverage. However, based on reasonable assumptions about security of the attestation architecture, we show that $\mathcal{M}al$ is detectable with significant probability.

Furthermore, since individual attestation instances are independent, the compound probability for $\mathcal{M}al$ to evade detection can be made negligible. As discussed in Section 6 below, although this increased level of security comes at the cost of running several measurements, no additional hardware features are needed. This results in **the first** low-cost and secure hybrid attestation technique that has no impact on $\mathcal{P}rv$'s availability during attestation.

Shuffled (or random) memory coverage has been already suggested in the context of software-based attestation. However, it was done differently from *SMART*, in several ways. First, random coverage of memory in software-based attestation is not secret, i.e., $\mathcal{M}al$ is fully aware of the sequence of memory blocks traversal. In contrast, *SMART* assumes secrecy of this traversal pattern (shuffling), since it is generated based on $\mathcal{V}rf$'s one-time challenge and a secret key shared by $\mathcal{V}rf$ and $\mathcal{P}rv$, which is inaccessible to $\mathcal{M}al$, as part of the underlying SMART architecture. Also, as described in [15], [14], [13], memory blocks are measured several times before all are processed at least once. This redundant coverage is likely due to size restrictions and non-optimizable constraints of AttC .

In the rest of this paper, in the context of $\mathcal{P}rv$ implementing *SMART*, we analyze different evasion strategies based on roving $\mathcal{M}al$'s varying degree of knowledge about the progress of shuffled measurements.

3. *SMART*: Model and Assumptions

We assume that $\mathcal{P}rv$'s memory is divided into n blocks M_1, \dots, M_n . We require $\mathcal{P}rv$ to conform to the SMART architecture, as described in [7] and summarized above, augmented by an anti-DoS extension proposed in [3], which requires $\mathcal{P}rv$ to maintain either a reliable read-only clock (RROC) or a monotonic secure counter. This is needed to authenticate and detect replayed or reordered $\mathcal{V}rf$'s attestation requests. In addition, *SMART* entails one important change with respect to SMART:

We relax the atomicity requirement of SMART such that the measurement process implemented by AttC can be interrupted after it measures each memory block.

Let σ be a permutation randomly selected for a given attestation instance (measurement) of M . That is, block $M_{\sigma(i)}$ is measured at step i . Let t_1, \dots, t_n be the times at which blocks $M_{\sigma(1)}, \dots, M_{\sigma(n)}$ are measured, respectively.

Let M^* be M in benign state. Let $R^* = F_K(M^*)$ be the measurement corresponding to the healthy state, computed by the measurement routine F using key K . Finally, let R be the measurement actually computed over M in a given attestation instance².

2. Since the measurement process takes a non-negligible amount of time and since M can change during that time, we can not write $R = F(M)$.

We define a *benign measurement* as the event $R = R^*$. We further define the probability of \mathcal{Mal} evading detection with strategy S as:

$$P_S = \Pr(R = R^* \mid M \neq M^*)$$

We assume that measurement of each block M_i is atomic, i.e., uninterrupted. This can be guaranteed by the attestation architecture, for instance by disabling/re-enabling interrupts at the start/end of the measurement of a block. (This is in contrast to SMART where the entire process is atomic.) Potential interruptions by other tasks running on \mathcal{Prv} must thus be scheduled between the measurement of two blocks. Let t_{\max} denote the maximum non-interruptibility interval. The size of a memory block M_i is thus set such that the time to measure it is at most t_{\max} .

4. Roving \mathcal{Mal} Evasion Strategies

Since \mathcal{Mal} 's goal is to avoid detection, it must restore each block where it resides to a benign state before that block is measured. This section considers the optimal strategy for \mathcal{Mal} , under various assumptions about its capabilities and knowledge, and the associated probability of detection.

We initially assume that \mathcal{Mal} occupies a single block. The case where it resides in multiple blocks is discussed later in Section 5.3. Without loss of generality, we also assume that \mathcal{Mal} is active during the entire attestation process, and can thus interrupt it and make changes to any block of M at any point as long as it does so between the intervals of attestation process measuring a single block. Reactive \mathcal{Mal} , and \mathcal{Mal} with restricted number of interruptions are discussed in Sections 5.4 and 5.5, respectively.

4.1. Erasure

One trivial evasion strategy for \mathcal{Mal} is to simply erase itself as soon as possible, perhaps to re-infect \mathcal{Prv} at a later time. Assuming that \mathcal{Mal} is aware of the incoming attestation request from \mathcal{Vrf} , or it interrupts the attestation process before it starts (or early on during its execution), erasure seems difficult, if not impossible, to mitigate. In the rest of this paper, we focus on strategies whereby \mathcal{Mal} attempts to remain on \mathcal{Prv} while evading detection.

4.2. Relocation Techniques

Clearly, if \mathcal{Mal} remains where it is, it can not escape detection. Otherwise, it must relocate itself, at least once. We identify and explore three intuitive \mathcal{Mal} flavors (which vary in the degree of knowledge) and their associated probabilities of successful evasion. As mentioned earlier, we assume below that \mathcal{Mal} occupies a single memory block.

4.2.1. Knowledge of Future Volume (KFV). During attestation, the volume (size) of memory that has not yet been measured is the least amount of actionable information

that \mathcal{Mal} might have. This knowledge can be acquired by measuring the time elapsed since the start of attestation and estimating the number of memory blocks already measured. This is based on a realistic assumption that \mathcal{Mal} is aware of: (1) time when attestation began, and (2) time to measure one memory block. We refer to this degree of knowledge as the KFV model.

Theorem 1. *The optimal strategy for KFV \mathcal{Mal} is to relocate after every memory block is measured. It would thus move a total of $n-1$ times, assuming that it can interrupt the attestation process at each block boundary. The probability of evasion is:*

$$P_{FV} = \left(1 - \frac{1}{n}\right)^n \approx e^{-1} \approx 0.37$$

Proof. Let M_{m_i} denote the block containing \mathcal{Mal} at t_i , for $1 \leq m_i \leq n$. Having the ability to interrupt the attestation process between t_i and t_{i+1} , \mathcal{Mal} can either stay put or relocate. Let p_i be the probability of \mathcal{Mal} getting ‘‘caught’’ exactly at t_{i+1} :

$$p_i = \Pr(m_{i+1} = \sigma(i+1) \mid m_k \neq \sigma(k), 1 \leq k < i),$$

for $1 \leq i < n$.

If \mathcal{Mal} relocates, two outcomes may occur: either (1) its new location $M_{m_{i+1}}$ has been already measured (there are i such blocks), in which case it will certainly not be caught, or (2) $M_{m_{i+1}}$ was not measured yet (there are $n-i$ such locations), in which case it will be caught with probability $\frac{1}{n-i}$. Consequently:

$$p_i^{\text{move}} = \frac{i}{n} \cdot 0 + \frac{n-i}{n} \cdot \frac{1}{n-i} = \frac{1}{n}. \quad (1)$$

If \mathcal{Mal} does not relocate, two situations may occur. Let j be the last interval when \mathcal{Mal} moved. (If it never moved, $j = 0$.) Again, $M_{m_{i+1}} = M_{m_{j+1}}$ might have been already measured, in which case \mathcal{Mal} will not be caught. This occurs with probability $\frac{j}{n}$. Indeed, since $M_{m_{j+1}}$ can not have been measured in the last $i-j$ steps (since we assume \mathcal{Mal} has not been detected so far), it must have been measured in the j first ones.

Otherwise, if $M_{m_{j+1}}$ has not been measured yet (which occurs with probability $1 - \frac{j}{n}$), \mathcal{Mal} will be caught with probability $\frac{1}{n-i}$ (for the same reason as in Eq. (1)). Consequently,

$$p_i^{\text{stay}} = \frac{j}{n} \cdot 0 + \frac{n-j}{n} \cdot \frac{1}{n-i} = \frac{n-j}{n(n-i)}$$

If $j < i$, $p_i^{\text{stay}} > p_i^{\text{move}}$. Therefore, it is always beneficial for \mathcal{Mal} to relocate. Interestingly, the new location $M_{m_{i+1}}$ does not matter, as long as $m_{i+1} \neq m_i$. Relocation at each interval leads to an overall evasion probability of:

$$P_{FV} = (1 - p_i)^{n-1} \cdot \Pr(m_1 \neq \sigma(1)) = \left(1 - \frac{1}{n}\right)^n$$

$\Pr(m_1 \neq \sigma(1))$ is the probability that the first block measured is not M_{m_1} . The approximation to e^{-1} results from the limit definition of e . \square

4.2.2. Knowledge of Future Coverage (KFC). In addition to knowing *how many* blocks have been measured, \mathcal{M} al might also learn *which* blocks have been measured. That is, after t_i , \mathcal{M} al knows $\{\sigma(1), \dots, \sigma(i)\}$. Based on this, \mathcal{M} al can infer $\{\sigma(i+1), \dots, \sigma(n)\}$. This is different from knowing precise values $\sigma(i+1), \dots, \sigma(n)$; see Section 4.2.3 below. We refer to this as the KFC model.

This greater knowledge might stem from a side-channel vulnerability in the implementation of the attestation process, e.g., if measured blocks can be distinguished from non-measured ones from the point of view of \mathcal{M} al. Alternatively, it could be the result of the attestation process carelessly storing $\{\sigma(1), \dots, \sigma(i)\}$ (See Section 7).

Theorem 2. *The optimal strategy for KFC \mathcal{M} al is to change its location to $M_{\sigma(1)}$ immediately after t_1 . The probability of evasion is:*

$$P_{FC} = 1 - \frac{1}{n}$$

Proof. Since $\sigma(1)$ is unknown before t_1 , \mathcal{M} al can not make any informed decision, and thus would be discovered if $m_1 = \sigma(1)$. After t_1 , \mathcal{M} al knows that $M_{\sigma(1)}$ was measured. Thus, it can safely relocate there and remain until the end. The result follows trivially: $\Pr(m_1 = \sigma(1)) = \frac{1}{n}$. \square

4.2.3. Knowledge of Future Order (KFO). In addition to future coverage, \mathcal{M} al might know $\sigma(i+1), \dots, \sigma(n)$, and thus also $\{\sigma(1), \dots, \sigma(i)\}$. Whether \mathcal{M} al also knows the past order of coverage $(\sigma(1), \dots, \sigma(i))$ is irrelevant; see Section 5. \mathcal{M} al might acquire this additional knowledge due to $\sigma(i+1), \dots, \sigma(n)$ being insecurely stored by the attestation process; see Section 5. The leakage can also occur due to some vulnerability in the random block selection process, e.g., in case of a weak random number generation. We refer to this as the KFO model.

Theorem 3. *The optimal strategy for KFO \mathcal{M} al is to relocate to any M_j with $j \neq \sigma(1)$ before t_1 if $M_{\sigma(1)}$ is its initial location, and then to $M_{\sigma(1)}$ immediately after t_1 . The probability of evasion is:*

$$P_{FO} = 1$$

Proof. If \mathcal{M} al is located in $\sigma(1)$ (the next block to be measured), it must relocate. After t_1 , \mathcal{M} al knows that $M_{\sigma(1)}$ was measured, so it can safely relocate there and remain to the end. Consequently, \mathcal{M} al must move once (or twice, if $M_{\sigma(1)}$ is its initial location) in order to evade detection. \square

5. Discussion

We now discuss the viability of aforementioned malware flavors and several related issues.

5.1. Realistic Scenarios

We believe that KFV is quite realistic. It requires little insight and sophistication from \mathcal{M} al. It is viable even with a secure hybrid attestation framework and a secure handling

of σ . As discussed in Section 5.2, even when \mathcal{M} al's timing estimates are inaccurate, it can attain the probability of evasion close to e^{-1} . In contrast, KFC and KFO \mathcal{M} al flavors are more far-fetched, as long as σ is placed in secure storage, or stored elsewhere in an encrypted and authenticated form; see Section 7.³ Therefore, the rest of this paper focuses on KFV \mathcal{M} al.

5.2. Unknown Timing

Recall that KFV \mathcal{M} al is assumed to know (t_1, \dots, t_n) . This may not always be the case in practice. For instance, other active processes might distort the timing, or spurious delays might be introduced by the AttC, as a potential countermeasure.

One possible strategy for \mathcal{M} al is to relocate as often as possible. As follows from the proof of Theorem 1, as long as \mathcal{M} al relocates at least once between two blocks being measured, the predicted probability of successful evasion is reached. While precise values for (t_1, \dots, t_n) may not be known, \mathcal{M} al can easily determine the lower bound on the time to measure one block. If \mathcal{M} al relocates with a frequency of that lower bound, it can attain the probability of evasion of e^{-1} .

5.3. \mathcal{M} al in Multiple Blocks

Despite the earlier assumption, \mathcal{M} al might not fit in a single memory block. Let s denote the number of blocks occupied by KFV \mathcal{M} al. Its evasion probability can be viewed as evasion by s separate single-block pieces of malware, with the probability of:

$$P_{FV}(S = s) = P_{FV}(S = 1)^s \approx e^{-s}.$$

5.4. Active vs Reactive Malware

So far, we assumed that \mathcal{M} al is active throughout the attestation process. We now broaden this to include \mathcal{M} al which wakes up (becomes active) at some point during attestation. Let w be the number of memory blocks measured thus far. Let $P_{\text{active}}(n, s)$ be the probability of evasion of an active \mathcal{M} al of size s for a given strategy on measurement of n blocks. The probability $P_{\text{reactive}}(n, s, w)$ of \mathcal{M} al following the same strategy reacting after w blocks is:

Theorem 4.

$$P_{\text{reactive}}(n, s, w) = \frac{\binom{n-s}{w}}{\binom{n}{w}} P_{\text{active}}(n-w, s)$$

Proof. The $\frac{\binom{n-s}{w}}{\binom{n}{w}}$ factor is the probability of none of \mathcal{M} al's s blocks being chosen in first w measurements. It is a special case of a hypergeometric distribution. The second factor is the probability that an active \mathcal{M} al escapes detection, with $n-w$ blocks remaining. \square

³ However, KFC or KFO might be possible if faulty AttC leaks information about σ , or if σ is leaked via some other side-channel means.

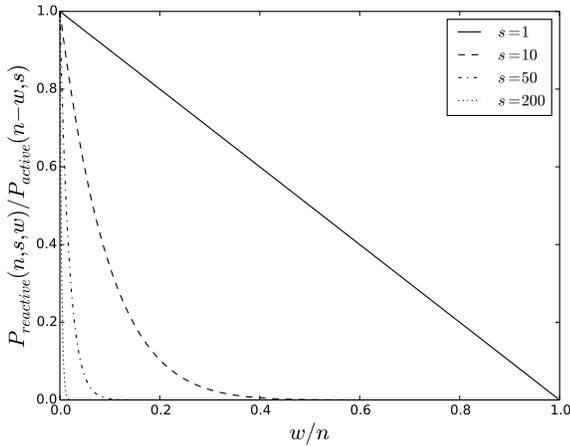


Figure 1: Effect of reactive malware.

It follows trivially that $P_{\text{reactive}}(n, s, 0) = P_{\text{active}}(n, s)$. Figure 1 shows the effect of $\mathcal{M}\text{al}$ that is only active after w steps.

5.5. Limited # of Interruptions

We now consider the case when KFV $\mathcal{M}\text{al}$ is limited to at most k interruptions during the entire attestation process instance. This might be motivated by $\mathcal{M}\text{al}$ aiming to reduce its timing footprint. Indeed, one possible countermeasure for the attestation process (with SMARM) that suspects roving $\mathcal{M}\text{al}$ presence is to measure its total elapsed (wall-clock) time. If it diverges significantly from the expected time, and if there is no legitimate justification, it can be an indication of activity by frequently relocating $\mathcal{M}\text{al}$.

Theorem 5. *The optimal strategy for KFV $\mathcal{M}\text{al}$ limited to k interruptions is to change its location after each group of $n/(k+1)$ blocks is measured. It thus moves k times and its probability of evasion is:*

$$P_{\text{FV}}(K = k) = \left(1 - \frac{1}{k+1}\right)^{k+1}$$

Proof. See Appendix A. \square

6. Reliable Detection

Using SMARM , the probability that one measurement does not detect $\mathcal{M}\text{al}$ presence is non-negligible. However, if multiple measurements are taken, the overall false negative probability decreases exponentially. Multiple measurements can be obtained via: (1) independent consecutive attestation instances, (2) batch mode, e.g. $\mathcal{V}\text{rf}$ sends m challenges at once and receives m measurements, or (3) self-measurements by $\mathcal{P}\text{rv}$ itself, as described in [4]. Given m independent measurements, each with probability P of a false negative, the overall false negative probability is $P_m = P^m$. That is, with $P = e^{-1}$ this gives, e.g., $P_7 < 10^{-3}$, and $P_{13} < 10^{-6}$.

7. Block Permutation in Practice

This section discusses a trial implementation of random block shuffling and its security implications.

7.1. Permutation Computation and Storage

Recall that the random one-time permutation σ is computed at the start of the attestation process, prior to measuring any memory blocks. One way to compute it efficiently is by using the well-known *Fisher-Yates* [8] (also known as *Knuth's* [11]) shuffle method.

$\mathcal{P}\text{rv}$'s PRNG is seeded with $H(c, K)$ where c is $\mathcal{V}\text{rf}$'s challenge, K is the key securely stored by $\mathcal{P}\text{rv}$ (as part of SMART), and H is a hash function. As mentioned at the start of Section 3, we assume that $\mathcal{P}\text{rv}$ authenticates each attestation request and, as part of that process [3], establishes its freshness, i.e., detects replay attacks. Thus, once an attestation request is validated, $\mathcal{P}\text{rv}$ knows that c (contained therein) is guaranteed to be unique; hence, $H(c, K)$ is unique as well. This guarantees that random values σ_i produced by PRNG are both fresh and secret, i.e., unpredictable by $\mathcal{M}\text{al}$.

Once computed, $\sigma = \{\sigma(1), \dots, \sigma(n)\}$ is stored in secure memory. The underlying security architecture ensures that (similar to K), σ can be written and read only by AttC . This requires $n \lceil \log n \rceil$ bits of storage. When the measurement of block $M_{\sigma(i-1)}$ is completed (or when attestation starts at $i = 1$), $\sigma(i)$ is read and $M_{\sigma(i)}$ is fed to the MAC function. Once all blocks are measured, σ remains in secure storage. It is then over-written at the start of the next attestation instance.

If the size of the additional secure storage is a concern, the following variant is an alternative. The permutation σ is instead stored encrypted and authenticated in *insecure storage*, and no additional secure storage is required. Each index $\sigma(i)$ is encrypted individually as a block of the block cipher. Since efficiency is a concern, the block cipher operates in CTR mode [5], so that random access to individual $\sigma(i)$ is possible. The counter is set to a concatenation of the $\mathcal{V}\text{rf}$ one-time challenge c and i . The encryption of $\sigma(i)$ is thus $E_K(c||i) \oplus \sigma(i)$.

In addition, a MAC of the encrypted $\sigma(i)$ is concatenated (cf. Encrypt-then-Mac [9]). A scenario where $n > 2^{128}$ being unrealistic, using a block cipher and MAC with a block size of 128 is sufficient. For instance, using AES-128 as E and a SHA-2-based HMAC means that σ is stored on $48n$ bytes. At the start of the measurement of each block, the encrypted $\sigma(i)$ is decrypted, its MAC verified, and the corresponding block read.

This variant trades off secure storage for regular storage and increased computation cost (see experiment results in Section 8).

7.2. Memory Overhead

We estimate memory overhead for storing σ on I.MX6-SabreLite [1], a popular and inexpensive development board representative of low- to medium-end IoT devices.

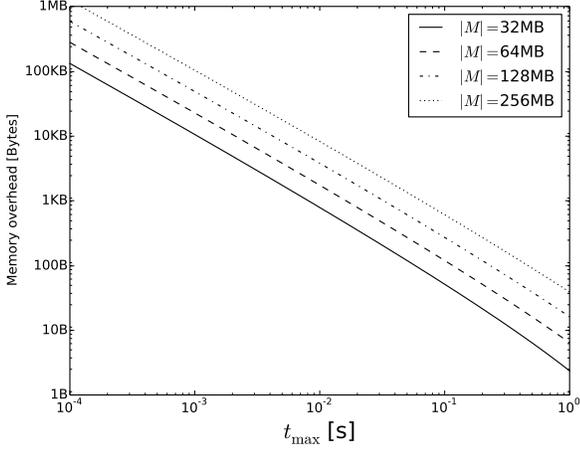


Figure 2: Memory requirement for σ on a I.MX6-SabreLite, as a factor of t_{\max} and $|M|$.

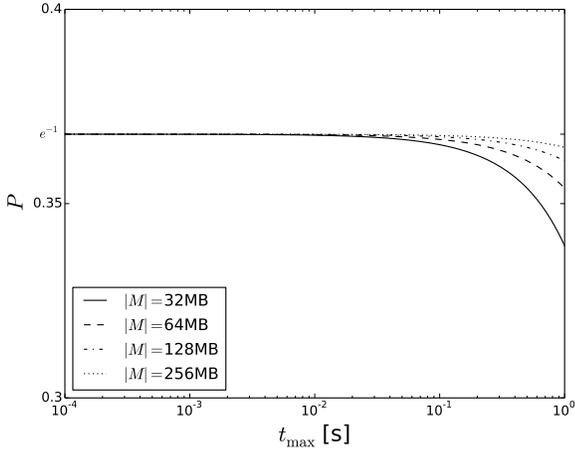


Figure 3: Probability of evasion for KfV $\mathcal{M}al$, on a I.MX6-SabreLite, as a factor of t_{\max} and $|M|$. $\mathcal{M}al$ is proactive and has is of size $s = 1$.

As discussed in Section 3, the maximum non-interruptibility interval t_{\max} influences block size and their number. Given total memory size $|M|$ and throughput θ $n = \frac{|M|}{\theta t_{\max}}$. This allows us to predict the amount of storage for σ . Figure 2 shows this overhead, for varying values of t_{\max} and $|M|$, with a throughput of $\theta \approx 20.48\text{MB/s}$.

Figure 3 shows the probability of evasion for KfV $\mathcal{M}al$. It shows that a higher t_{\max} decreases the chances of escaping detection. As evident from Figure 2, it also requires less storage. However, in order to guarantee good availability on $\mathcal{P}rv$, t_{\max} should not be too high.

8. HYDRA Implementation

8.1. Overview

We implemented *SMARM* in the context of HYDRA [6] on an I.MX6-SabreLite. HYDRA implements a hybrid RA design for devices with a Memory Management Unit (MMU). It builds upon the formally verified seL4 [10] microkernel, which ensures process memory isolation and enforces access control to memory regions. Using the (mathematically) proven isolation features of seL4, access control rules can be implemented in software and enforced by the microkernel.

HYDRA stores K and $\mathcal{A}ttC$ in a writable memory region and configures the system such that no other process, besides the measurement process, can access these memory regions. Access control configuration in HYDRA also involves the measurement process having exclusive access to its thread control block as well as to memory regions used for key-related computations. The measurement process starts with the highest maximum controlled priority (MCP). This ensures execution of the measurement process will run uninterruptedly as long as its “priority” remains the highest. Note that, in seL4, a priority is the effective priority of a process, which can be increased and decreased at run-time as long as it does not exceed its MCP value. In contrast, a process cannot increase its MCP after it is set (but a decrease is possible).

The formally verified version of seL4 uses a simple preemptive round-robin scheduler. Time is divided equally in timeslices, and each timeslice is assigned a given process to run, based on its priority.

The measurement procedure in the HYDRA-based *SMARM* implementation is as follows. At the start of a timeslice, the measurement process is set to highest priority (so that it can not be interrupted). Once the measurement of the block is done (this takes t_{\max} seconds), the measurement process decreases its priority and yields (via the `seL4_Yield` system call) its remainder of the timeslice. This effectively allows another process (which may include $\mathcal{M}al$) to run immediately after each block’s measurement with a fresh timeslice. Once the next timeslice of the measurement process begins, the next block is measured ⁴.

8.2. Experimental Results: *SMARM* with/without Secure Storage

We generate the random permutation σ using the Fisher-Yates shuffle method, discussed in Section 7. The PRNG is implemented using AES-256 in CTR Mode, seeded with $\text{SHA-256}(c, K)$. In the variant where secure storage is not available, we implement the underlying block cipher and MAC function as AES-128 and SHA-256-based HMAC respectively.

4. Technically, each interrupting process besides the measurement process is allowed one timeslice (round-robin scheduling). However, for simplicity we assume a single (possibly infected) such process carrying out $\mathcal{P}rv$ ’s main task.

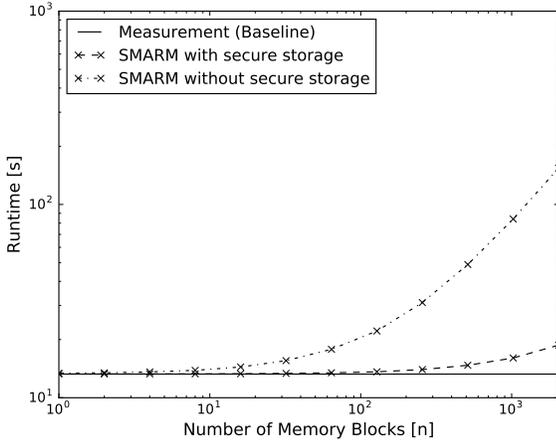


Figure 4: Runtime of *SMARM* with/without secure storage for σ as a factor of n . $|M|$ is fixed to 256MB. It is also assumed that no other process exists besides the measurement process.

Figure 4 shows overall runtime of these two variants (i.e. *SMARM* with/without secure permutation storage) compared to that of the baseline measurement. With enough secure storage, *SMARM* incurs 41% overhead over the baseline measurement with $n = 2048$ (or $t_{\max} = 0.006s$). This overhead decreases as n decreases; it achieves 0.8% at $n = 32$ (or $t_{\max} = 0.4s$). On the other hand, without secure storage for σ , *SMARM* spends significant amount of time performing additional encryptions and MAC computations. The overhead in this case can be as high as 10,636% (at $n = 2048$) and as low as 2% (at $n = 1$). Thus, it might be more beneficial to deploy a lower number of memory blocks if a device does not have room for secure storage.

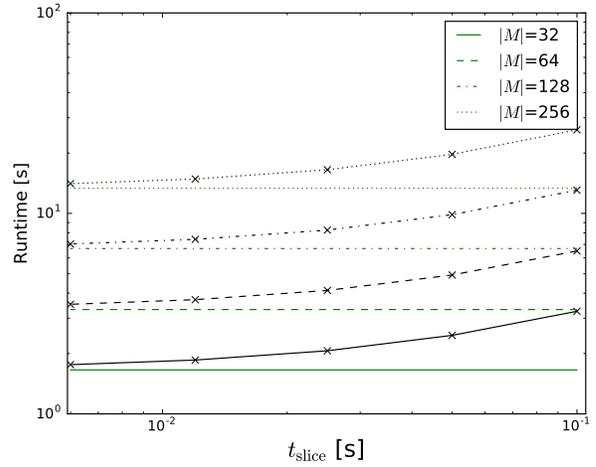
8.3. Experimental Results: Different t_{slice} and t_{\max}

Figure 5 illustrates the worst-case runtime of *SMARM* on this HYDRA implementation, compared to the uninterrupted measurement of M , as a factor of t_{slice} , t_{\max} , and $|M|$. The overall *SMARM* runtime: (1) increases as t_{slice} increases, and (2) decreases as t_{\max} increases. We found that the overhead can be up to a factor of $1 + \frac{t_{\text{slice}}}{t_{\max}}$, while another process can run for up to t_{slice} seconds every $t_{\text{slice}} + t_{\max}$ seconds.

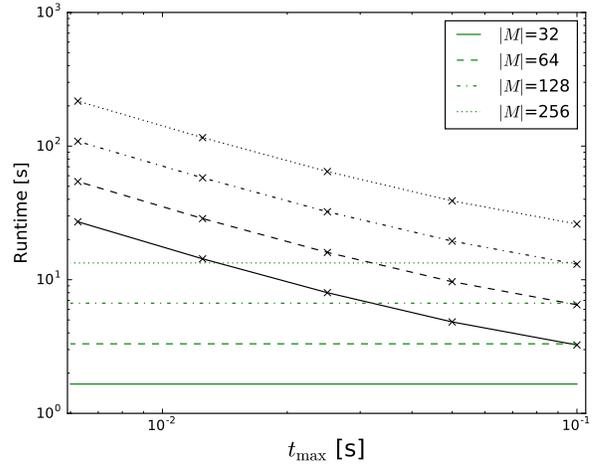
9. Conclusion

This paper presents *SMARM*, an attestation technique based on measuring $\mathcal{P}rv$'s memory in a random and secret order.

Contrarily to most hybrid attestation schemes, the measurement process can be interrupted and the attested memory modified. This makes attestation having no practical impact on the availability of $\mathcal{P}rv$ for its main operation. This is a requirement in time- and/or safety-critical applications.



(a) Runtime as a factor of t_{slice} and $|M|$. t_{\max} is fixed to 100ms.



(b) Runtime as a factor of t_{\max} and $|M|$. t_{slice} is fixed to 100ms.

Figure 5: Runtime comparison between *SMARM* (crosses, black) and baseline measurement (green) with different t_{slice} , t_{\max} and $|M|$; it is assumed that a single process exists besides the measurement process.

Roving malware normally present a threat to interruptible measurement over malleable memory. However, since in *SMARM* the memory is measured in unpredictable and secret order, malware can not make informed decisions to escape detection with certainty.

Using a secure framework, such as HYDRA, allows to keep the permutation unknown to potential malware. This caps the evasion probability to $e^{-1} \approx 37\%$. Since each attestation session is independent, the probability that a piece of malware present on $\mathcal{P}rv$ consistently escapes detection after multiple attestation sessions drops exponentially. After 13 checks, that probability is below 10^{-6} .

SMARM thus results in: (1) increased time to attain a negligible probability of false negatives, and (2) additional memory to store the current permutation, as compared to

techniques using deterministic coverage order with non-interruptibility or non-malleability. However, to the best of our knowledge, *SMARM* is the only RA technique that mitigates roving malware for time- and/or safety-critical applications.

References

- [1] Boundary Devices. i.mx6 arm development board. <https://boundarydevices.com/product/sabre-lite-imx6-sbc/>.
- [2] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. TyTAN: tiny trust anchor for tiny devices. In *ACM/IEEE Design Automation Conference (DAC)*, 2015.
- [3] Ferdinand Brasser, Ahmad-Reza Sadeghi, and Gene Tsudik. Remote attestation for low-end embedded devices: the prover’s perspective. In *ACM/IEEE Design Automation Conference (DAC)*, 2016.
- [4] Xavier Carpent, Norrathep Rattanavipanon, and Gene Tsudik. Erasmus: Efficient remote attestation via self-measurement for unattended settings. *arXiv preprint arXiv:1707.09043*, 2017.
- [5] Whitfield Diffie and Martin E Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, 1979.
- [6] Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Hydra: Hybrid design for remote attestation (using a formally verified microkernel). In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec ’17, pages 99–110, New York, NY, USA, 2017. ACM.
- [7] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. In *Network and Distributed System Security Symposium (NDSS)*, 2012.
- [8] Ronald Aylmer Fisher, Frank Yates, et al. Statistical tables for biological, agricultural and medical research, edited by ra fisher and f. yates, 1963.
- [9] ISO/IEC. 19772:2009, Information technology – Security techniques – Authenticated encryption, February 2009.
- [10] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.
- [11] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.
- [12] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. TrustLite: A security architecture for tiny embedded devices. In *ACM European Conference on Computer Systems (EuroSys)*, 2014.
- [13] Yanlin Li, Jonathan M McCune, and Adrian Perrig. Viper: verifying the integrity of peripherals’ firmware. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 3–16. ACM, 2011.
- [14] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. *ACM SIGOPS Operating Systems Review*, December 2005.
- [15] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. SWATT: Software-based attestation for embedded devices. In *IEEE Symposium on Research in Security and Privacy (S&P)*, 2004.
- [16] Jaikumar Vijayan. Stuxnet renews power grid security concerns, june 2010.

Appendix

Proof of Theorem 5. Theorem 1 showed that relocating is always preferable for $\mathcal{M}al$ to remaining in the same block. What remains to determine is when to optimally make these interruptions.

Let n_1, \dots, n_k be the number of blocks measured before each interruption, and let $n_{k+1} = n - \sum_{i=1}^k n_i$ be the remaining blocks after the last interruption. The probability for $\mathcal{M}al$ to escape detection is thus:

$$P = \prod_{i=1}^{k+1} \left(1 - \frac{n_i}{n}\right)$$

Let P be the probability pertaining to the strategy n_1, \dots, n_k . Let an alternative strategy n'_1, \dots, n'_k (and the corresponding probability P') be defined as:

$$n'_i = \begin{cases} n_a + c & \text{if } i = a \\ n_b - c & \text{if } i = b \\ n_i & \text{else,} \end{cases}$$

for $a \neq b$: $n_a \geq n/(k+1)$, $n_b \leq n/(k+1)$. Such a pair is guaranteed to exist because $n_{k+1} = n - \sum_{i=1}^k n_i$. We thus have:

$$\begin{aligned} \frac{P}{P'} &= \frac{\left(1 - \frac{n_a}{n}\right) \left(1 - \frac{n_b}{n}\right)}{\left(1 - \frac{n_a+c}{n}\right) \left(1 - \frac{n_b-c}{n}\right)} \\ &= \frac{\left(1 - \frac{n_a}{n}\right) \left(1 - \frac{n_b}{n}\right)}{\left(1 - \frac{n_a}{n}\right) \left(1 - \frac{n_b}{n}\right) + \frac{c}{n} \left(\frac{n_b}{n} - \frac{n_a}{n}\right) - \frac{c^2}{n^2}}. \end{aligned}$$

Since $n_a \geq n_b$, the denominator is smaller than the numerator, and thus $P \geq P'$. This shows that any strategy that diverges from $n_i = n/(k+1)$ will be sub-optimal. \square