

# SimHAT: Simulated Hardware-based Attestation evaluation Tool

Kirti Chawla, Taniya Siddiqua

Department of Computer Science  
School of Engineering and Applied Science  
University of Virginia  
Charlottesville, Virginia, 22904-4740  
{kirti, ts7au}@cs.virginia.edu

## Abstract

*Embedded devices have come to play an important role in our daily lives. Increasingly, diverse fields like healthcare systems, banks and more are using these devices to automate underlying complex tasks. This reliance has led to a significant development – exposure of devices to malicious attacks, including control and subversion of software onboard. Software attestation has been heralded as a solution to minimize the effects of malicious attacks on the devices. With low power, relatively small memory size and less processor frequency, standard software attestation techniques cannot be directly applied to embedded devices. To solve this problem, we propose a novel design of simulated hardware-based tool. Our tool can be used to evaluate competing attestation techniques. Furthermore, We present a reference implementation and evaluate the runtime cost of a home-grown attestation procedure.*

**Keywords:** Attestation, Embedded devices, Performance Evaluation

## I. Introduction

Today, we witness the use of large amount of computing devices in our day to day lives. Together these devices form a system, which is used to provide various services, like healthcare, banks, security and more. Each such device is an embedded system, which offers limited processing and memory capabilities and is built for specific purpose. A concerted effort to control these devices for malicious purpose can have grave implications. A malicious control could be targeted at controlling the processors or injecting fatal code into memory or destroying critical components of

the device. To subvert this malicious control, the devices should be able to give express guarantees of integrity of various components to an entity within the device or to external entity. If the device is not able to provide these guarantees then it should cease to be part of system.

For the rest of paper, the scope of attack is limited to subversion or control attacks mounted on software of an embedded device.

Software attestation is a procedure, by which an external agent, called attester, is able to verify the contents of memory at runtime, infer the integrity of a device and provide guarantees of the attested devices to interested user.

Software attestation is proposed as the solution to minimize the effects of malicious attacks including code-injection, memory-control and more. Numerous software attestation techniques are present in verbatim, which provide the aforementioned features for point to point [1] or network based connection [4]. Other techniques use hardware in particular way to achieve the similar effect [2, 3]. It is believed that in near future new and optimal methods for performing software attestation will come in light.

Embedded devices, generally operate in constrained environment, involving less power, small memory size and low processor frequency and the cost of software attestation for large number of devices can entail significant penalty (in terms of person-hours).

Many off-the-shelf software attestation techniques cannot be directly applied to these devices due to the nature of constraints that are present. We believe that the software attestation community will invent new methods, keeping in mind the constraints.

Our goal is to evaluate competing software attestation techniques for the runtime cost, to draw qualitative inferences on underlying architecture and to

provide suggestions for optimizing a given setup used for performing software attestation.

We present a novel design based on simulated hardware model for evaluation of software attestation (Section II) called *SimHAT*. We demonstrate a proof-of-concept based experimental tool (Section III). Consequently, we evaluate and infer about underlying architecture for providing optimization suggestions (Section IV). Finally, we lay foundation for future work (Section V).

## II. System Model

In this section, we introduce the system preliminaries, design of software attestation evaluation tool based on simulated hardware model and a *home-grown* attestation procedure.

### A. System Preliminaries

In order to evaluate various software attestation procedures during design phase of development, simulation based techniques yield better cost-to-performance ratio over traditional methods.

Furthermore, this approach also shortens time-to-market for embedded devices consisting of pertinent software attestation feature. For the construction of our software attestation evaluation tool, we use a modeling infrastructure.

This infrastructure is called *MIDAS* [10] (or Modeling Infrastructure for Dynamic Active Storage). It simulates multiple instances of *SimpleScalar* [8] and *DiskSim* [9] simulators to form a framework of simulators. *SimpleScalar* is an architectural simulator, which has been developed at *University of Wisconsin, Madison*. With the help of this simulator, various design space factors can be explored. *DiskSim* is disk system simulator, which has been maintained by *Carnegie Mellon University*. It supports models of most secondary storage devices. It is capable of generating internal synthetic workloads.

By joining multiple instances of *SimpleScalar* and *DiskSim*, *MIDAS* facilitates simulation of simulators. It is both cycle-accurate and event-based. Furthermore, it supports homogeneous and heterogeneous multi-core as well as active disks. Our goal is to design *SimHAT* (a software attestation evaluation tool) based on *MIDAS*, which will aid in evaluating various software attestation procedures.

### B. Design

The design of *SimHAT* is involves 3 sub-modules viz. *AttestModule*, *CryptoModule* and *HelperModule*.

*AttestModule* provides the functionality of attestation to a designated Attester. *CryptoModule* is used by both Attester and Device. *HelperModule* provides useful functionality to both *AttestModule* and *CryptoModule*. The complete design is shown as under:

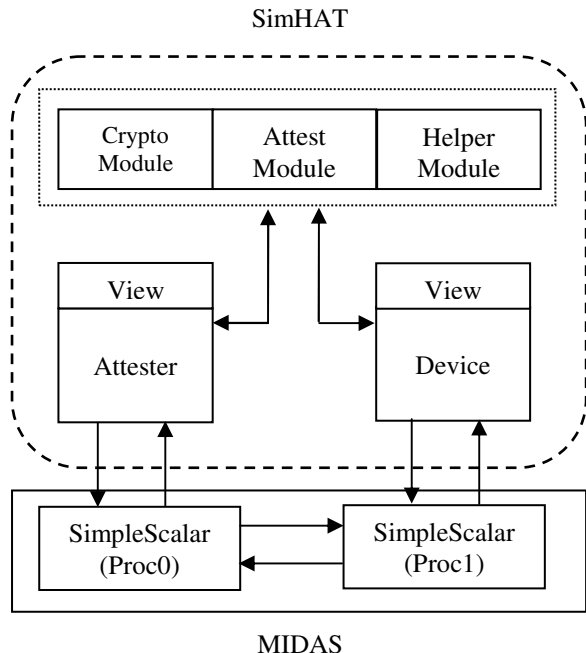


Fig. 1. Design of SimHAT on MIDAS

*CryptoModule* consists of pseudo-random stream generation algorithm and hash generation algorithm. Later in implementation section, we use RC4 [6] as pseudo-random stream generation algorithm and SHA1 [7] as hash generation algorithm. *CryptoModule* is designed in a way to facilitate addition of new software attestation procedures, which includes adding new algorithms for pseudo-random stream generation and hash generation. It is not mandatory to use only pseudo-random generation algorithm and hash generation algorithm to perform attestation, a software attestation procedure designer may differently. *CryptoModule* aids in that context as well.

An Attester uses *AttestModule* to perform attestation on a given Device. Internally, *AttestModule* utilizes the functionality of *CryptoModule* to achieve this effect. This completes the enumeration of building blocks of *SimHAT*.

### C. Attestation procedure

A software attestation procedure is complete only when, the generated hash is influenced by factors

including *Attester*-side view of memory, *Device*-side view of memory, challenge sent from *Attester* to *Device* and number of memory locations required for pseudo-random traversal. For the rest of the paper, we will assume that a software attestation procedure involves a variant of pseudo-random memory traversal. This can be seen as under:

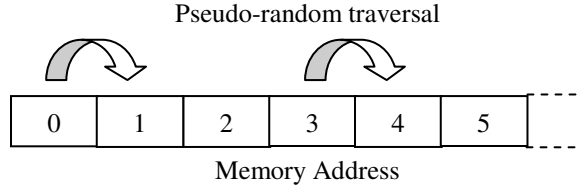


Fig. 2. Pseudo-random traversal on a window of memory

In our *home-grown* software attestation procedure, hash generation algorithm is dependent on memory view of Attester ( $V_A$ ), number of memory locations ( $C_A$ ), size of memory ( $M_A$ ) and challenge string ( $CH_A$ ). Formally, this can be seen as under:

$$\text{Attester-Side Hash} \rightarrow H_A = (V_A, C_A, M_A, CH_A)$$

$$\text{Device-Side Hash} \rightarrow H_D = (V_D, C_D, M_D, CH_D)$$

Given aforementioned semantics, the following protocol ensues between *Attester* and a given *Device*:

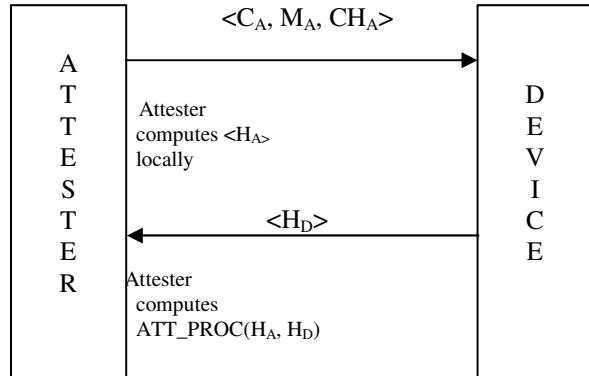


Fig. 3. Attestation protocol between Attester and Device

Formally, the outcome of software attestation by aforementioned method can be seen as under:

$$\text{ATT\_PROC}(H_A, H_D) = \begin{cases} \text{ATTEST}; > (H_A = H_D) \\ \text{UN-ATTEST}; > (H_A \neq H_D) \end{cases}$$

It should be note here that once *Attester* deems a *Device* as UN-ATTESTED, a custom action can be taken on *Device*-side to prevent any further harm by malicious code onboard. One such variant of action taken, is to put the *Device* in infinite loop. This concept is derived from a different application, wherein a wrong password leads to infinite loop [5].

Finally, we use RC4 as pseudo-random memory traversal algorithm and SHA1 as hash generation algorithm. The pseudo code of our software attestation procedure is shown as under:

```
Temp(I-1) = RotateLeft(Mem[Temp(I)] << 4) + PRAddr(I)
If (Temp(I-1) == 0x00) Temp(I-1) = PRAddr(I)
```

Fig. 4. Pseudo code of our software attestation procedure

### III. Implementation Details

The implementation consists of following set of files, which provide the functionality as described in previous sections.

```
.
|-- Makefile                : Makefile for building
|-- config.scm              : Multi-core configuration file
|-- sam.c                   : AttestModule
|-- sam.h                   : Header file of AttestModule
|-- scm.c                   : CryptoModule
|-- scm.h                   : Header file of CryptoModule
|-- sdefs.h                 : Standard definitions
|-- shm.c                   : HelperModule
|-- shm.h                   : Header file for HelperModule
|-- simhat.c                : SimHAT
```

Fig. 5. Source code view of SimHAT

The following illustration shows the snapshot of SimHAT in execution:

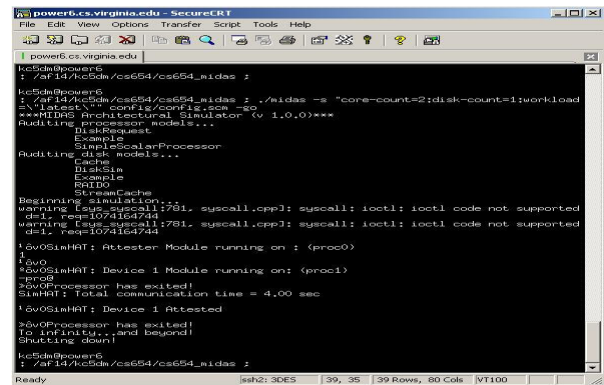


Fig. 6. Execution of SimHAT

## IV. Evaluation and Inferences

In order to evaluate the run-time cost of our home-grown software attestation procedure using SimHAT, experiments are run to measure variations in time and memory hierarchy depending on the attester and device frequencies.

### A. Results

We show results of evaluation of our home-grown procedure using SimHAT as under.

In order to evaluate, we are focusing on three factors as following: i) Attester-Device frequency pair  $\langle X, Y \rangle$  where  $X$  = Attester frequency and  $Y$  = Device Frequency, ii) Memory hierarchy  $\langle \text{IL1}, \text{DL1}, \text{UL2}, \text{Main Memory} \rangle$  and iii) View-Size of Attester and Device  $\langle A \rangle$ . We are varying these three factors and gathering various data to measure run-time cost and optimized setting.

#### a) Time Evaluation:

We gather time statistics depending on varying those three factors. First for a fixed memory model (IL1, DL1 = 32KB, UL2 = 256KB, Main Memory = 128 MB) we are keeping the Device frequency at 200MHz and varying the Attester frequency from 200MHz to 1000 MHz. From Fig.7 we can see that if we increase the ratio, it is taking more time to certify, especially to certify a device Un-Attested. So if we vary the frequency pair less, we will get better performance.

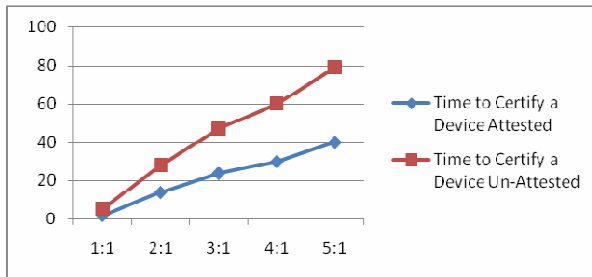


Fig. 7. Time Analysis depending on Attester-Device freq. ratio

Now, for arbitrary Attester-Device frequency pair, we vary the memory hierarchy. We are considering three types of memory model  $\langle \text{IL1}, \text{DL1}, \text{UL2}, \text{Main Memory} \rangle$  :  $\langle 16\text{KB}, 16\text{KB}, 128\text{KB}, 64\text{MB} \rangle$ ,  $\langle 32\text{KB}, 32\text{KB}, 256\text{KB}, 128\text{MB} \rangle$  and  $\langle 64\text{KB}, 64\text{KB}, 512\text{KB}, 256\text{MB} \rangle$ . From fig.8 we can see that, for  $\langle 32\text{KB}, 32\text{KB}, 256\text{KB}, 128\text{MB} \rangle$  memory model, we get the best performance. From this, we can infer that neither

small memory model nor large memory model helps in getting better performance.

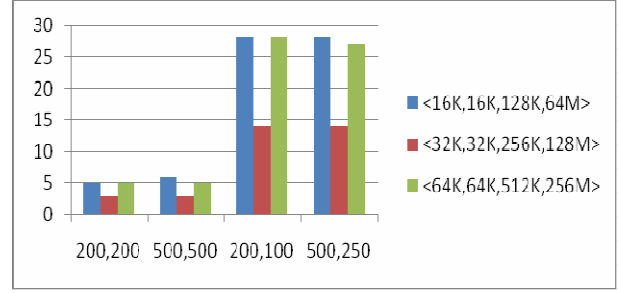


Fig.8. Time Analysis depending on Varying Mem. Model

Last case for timing analysis is depending on the variation of View-Size. We are considering three View-Sizes: 200KB, 100KB and 10KB. We can see from Fig.9 that even if View-Size is very less, it can attest correctly and we are getting the best performance. For a fixed View-Size,  $\langle 32\text{KB}, 32\text{KB}, 256\text{KB}, 128\text{MB} \rangle$  memory model is performing best. So we can achieve an optimized setting with less View-Size and  $\langle 32\text{KB}, 32\text{KB}, 256\text{KB}, 128\text{MB} \rangle$  memory model.

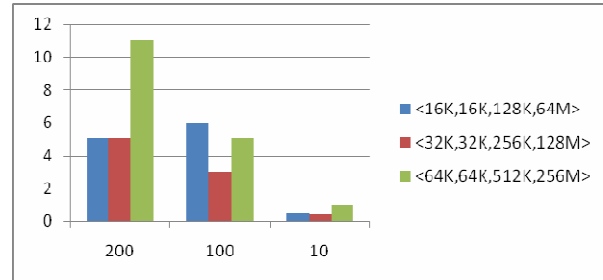


Fig.9. Time Analysis depending on View-Size

#### b) Memory Model:

If we consider the IL1 and DL1 miss ratios depending on varying Attester-Device Frequency pair and memory model, we can observe from Fig.10 and Fig.11 that for a fixed frequency pair, the varying memory model has almost no impact on miss ratios. From this we can come to a conclusion that in case of attestation, larger cache will not help.

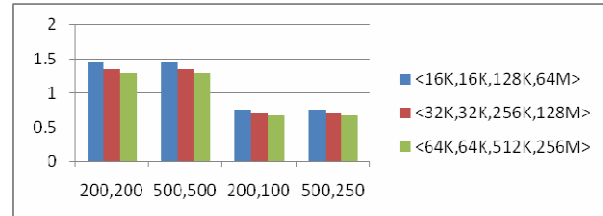


Fig.10. IL1 Miss Ratio Analysis(per thousand inst.) depending on varying freq. pair and mem model

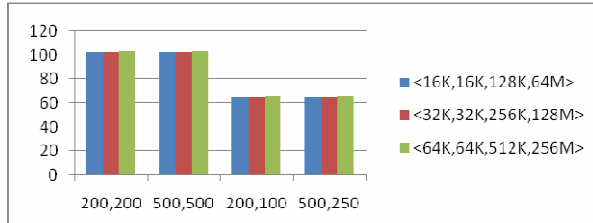


Fig.11. DL1 Miss Ratio Analysis(per thousand inst.) depending on varying freq. pair and mem model

## B. Inferences

From the above analysis, we can come to a conclusion that with a small View-Size, average Memory Model and less varying Attester-Device Frequency Pair, we can achieve the best performance for a given software attestation procedure.

## V. Future Work

Simultaneous multi-device attestation, more architectural inferences are some of the directions, which form the basis of future work.

## VI. Acknowledgments

We would like to express our gratitude towards Dr. Sudhanva Gurumurthi for stimulating discussions on architectural evaluation for software attestation and Dr. David Evans for answering numerous questions related to software attestation. Furthermore, we would like to express our gratitude towards, Clint Smullen, for providing insights on variable frequency setup for experiments and *MIDAS* in general.

## VII. References

- [1] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, Pradip Khosla, “*SWATT: SoftWare-based ATTestation for Embedded Devices*”, IEEE Symposium on Security and Privacy, 2004.
- [2] Xiaotong Zhuang, Tao Zhang, Hsien-Hsin S. Lee, Santosh Pande, “*Hardware Assisted Control Flow Obfuscation for Embedded Processors*”, Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2004.
- [3] Xiaotong Zhuang, Tao Zhang, Santosh Pande, “*HIDE: An Infrastructure for Efficiently Protecting Information Leakage on the Address Bus*”, Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2004.

[4] Rick Kennell, Leah H. Jamieson, “*Establishing the Genuinity of Remote Computer Systems*”, Proceedings of the 12th conference on USENIX Security Symposium, 2003.

[5] Xavier Boyen, “*Halting Password Puzzles: Hard-to-break Encryption from Human-memorable Keys*”, Proceedings of the 16th conference on USENIX Security Symposium, 2007.

[6] Ronald L. Rivest, “*RC4 – Stream Cipher*”, M. I. T, (<http://www.rsa.com/rsalabs/node.asp?id=2250>), 2007.

[7] P. Jones, “*RFC 3174 – US Secure Hash Algorithm 1 (SHA1)*”, 2001.

[8] Todd Austin, “*SimpleScalar Tutorial*”, University of Wisconsin-Madison, (<http://www.simplescalar.com>), 2004.

[9] Greg Ganger, Bruce Worthington, Yale Patt, “*DiskSim – The Disk Simulation Environment*”, Carnegie Mellon University, (<http://www.pdl.cmu.edu/DiskSim/>), 2007

[10] Clint Smullen, “*MIDAS Tutorial*”, University of Virginia, (<http://www.cs.virginia.edu>), 2007.