Turtles All The Way Down: Research Challenges in User-Based Attestation*

Jonathan M. McCune — *CMU/CyLab*
Adrian Perrig — *CMU/CyLab*
Arvind Seshadri — *CMU/CyLab*
Leendert van Doorn — *AMD*

A scientist once gave a public lecture describing how the Earth orbits around the sun and how the sun, in turn, orbits around the center of a collection of stars called our galaxy.

At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise."

The scientist gave a superior smile before replying, "What is the tortoise standing on?"

"You're very clever, young man, very clever," said the old lady, "but it's turtles all the way down!"

Abstract

Current trusted computing technologies allow computing devices to verify each other, but in a networked world, there is no reason to trust one computing device any more than another. Treating these devices as turtles, the user who seeks a trustworthy system from which to verify others quickly realizes that it's "turtles all the way down" because of the endless loop of trust dependencies. We need to provide the user with one initial turtle (the *iTurtle*) which is axiomatically trustworthy, thereby breaking the dependency loop. In this paper, we present some of the research challenges involved in designing and using such an *iTurtle*.

1 Introduction

Internet-connected computing devices feature complex software stacks which are often riddled with remotely-exploitable software vulnerabilities. Attackers can exploit these vulnerabilities to inject malware. The malware situation is made worse by the feature creep in software and the continuous introduction of new devices with vulnerable software on the Internet.

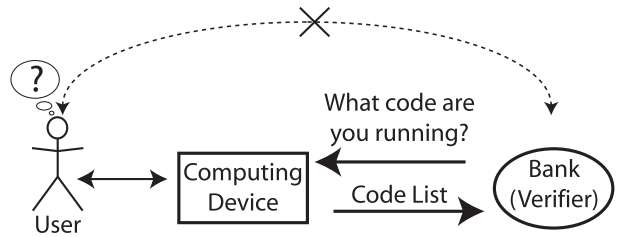

Figure 1: *Remote attestation does not create an authenticated channel between the verifier and the user. If verification fails, there is no way to inform the user. Thus, malware on the user's computing device can lie about verification results.*

In order to use their computing devices with confidence, users need to know if the software on their computing devices is infected by malware. The Trusted Computing Group (TCG) has proposed the mechanism of remote attestation to achieve this goal [14]. Remote attestation enables a remote *verifier* to learn the configuration of the software on any TCG-compliant computing device. The verifier can compare this configuration to a known-good configuration to detect deviations.

The TCG's remote attestation mechanism encounters at least three problems when we want to adapt it to user-based attestation. All three problems arise because the user does not have an axiomatically trustworthy device to perform verification. (1) The chain of trust created through the verification process does not propagate back to the user, because there is no authenticated channel between the user and the verifier (see Figure 1). (2) In a networked world, it is unclear to the user why the device that she uses as the verifier is any more trustworthy than her other devices. (3) Platform and user privacy are issues under TCG attestation. We present scenarios which will illustrate the above problems.

*This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and grant CCF-0424422 from the National Science Foundation, and equipment donations from AMD. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AMD, ARO, CMU, NSF, or the U.S. Government or any of its agencies.

Security-Sensitive Interactive Transactions. Consider a user who wants to perform online banking using her computer. If the bank's server is TCG-aware and the user's computer is TCG-compliant, then the bank's server can request an attestation from the user's computer to verify that the user's computer is running an approved software stack. If the server detects an unapproved software stack, then it can refuse service.

The problem with the above scenario is that there is no way for the bank's server to securely inform the user of the verification result (see Figure 1). If verification is unsuccessful, the user's computer cannot be trusted to display the correct result, since any notification mechanism that displays the verification result on-screen is vulnerable to spoofing. Malware installed on the user's computer could lie to the user that the attestation verified correctly at the bank, display a fake login page, and capture the user's login credentials.

There are two popularly suggested solutions to the problem of establishing an authenticated channel between the user and the bank: side-channels and trusted I/O. Automated side-channels use means of communication other than the user's computer to establish an authenticated channel. For instance, the bank's computer can send SMS messages or make telephone calls to convey the verification result to the user. However, any unauthenticated, automated side-channel may facilitate automated attack. Using a side-channel that is not automated, such as having a customer service representative make a phone call, is also not an option since the attacker can pretend to be a bank employee (a fact amply demonstrated by social engineering attacks). Besides, a non-automated side channel makes the verification expensive and potentially error prone by introducing additional human factors.

Upcoming trusted computing technologies, such as AMD's Presidio [1] and Intel's Trusted Execution Technology (TXT, formerly LaGrande) [5], which include hardware mechanisms for trusted I/O, do not solve the problem of establishing an authenticated channel between the user and the bank either. While they do include mechanisms for establishing trusted channels between platform components, they do not require any display to indicate that the channel is present [8]. Even if, in the future, these hardware technologies are extended to address this shortcoming, legacy systems will remain a problem.

The online banking example can be extended to any security-sensitive interactive transaction, such as remote login or e-commerce. One may also wish to consider how the problem presented in the above example can be applied to the examples in Chapter 2 of Balacheff et al. [3].

We believe that the best approach is for the user to use TCG load-time attestation to directly verify her own computer. This would allow the user to trust her computer (to the extent guaranteed by load-time attestation) to correctly display messages sent by the bank.

Specialized Computing Devices. Next, consider specialized computing devices such as 802.11 access points, home routers, GPS navigation systems, and printers. These devices may also contain information or perform tasks that users consider to be security-sensitive (e.g., the integrity of their map data, the secrecy of their printed documents, or the privacy of their photographs), making them attractive targets for malware.

It is unclear how to apply TCG-style remote attestation to these devices, because, in a networked world, the absence of an axiomatically trusted device means that the choice of which device to use as the verifier is not obvious. For example, it is unclear why the user should trust their cell phone to function as a verifier any more than they trust their laptop or desktop computer. Note also that today's attacks have moved "up" the software stack (e.g., cross-site scripting) and may apply to many device types. Furthermore, a third-party remote verifier cannot be used, even if one existed (no such verifiers exist today), since we re-encounter the problem of how to establish an authenticated channel between the user and verifier. Also, the user may not want to employ third-party verifiers due to privacy concerns.

Privacy Issues. During attestation, the verifier learns information about the challenged platform, including its TPM identity and software configuration. If the verifier is able to use any of this information to uniquely identify the challenged platform, then the verifier may be able to link the user to her platform (via, e.g., a single e-commerce transaction), thereby tracking the user's future interactions with that verifier.

The TCG has proposed Privacy CAs and Direct Anonymous Attestation (DAA) to anonymize the TPM used for attestations [14]. However, both of these schemes have complete privacy problems [4, 9]. Further, there are no known solutions to the problem of information leakage resulting from the inclusion of the software configuration of the platform in attestations. Property-based attestation is one proposed approach that tries to address the privacy issue of disclosing software configuration [11].

Enabling the user to verify her own computer would alleviate these privacy problems. Further, the user can set privacy policies on her computer with respect to remote attestations and trust the software on her computer to adhere to them (to the extent guaranteed by load-time attestation).

Contributions. So far, we have argued that providing the user with a device (the iTurtle) which is axiomatically trusted for verification purposes is a good way to address the shortcomings of TCG-style attestation. We further present some of the research challenges that arise when thinking about how to design and use such an iTurtle.

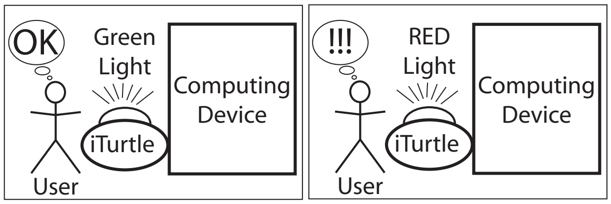

Figure 2: *Hypothetical scenario showing the use of the iTurtle. On the left, the user learns that her computing device is trustworthy. On the right, the user learns that her computing device has a problem.*

2 User-Observable Verification

We have discussed why current TCG attestation does not provide *user-observable verification.* Here, we conjecture how one might design a system for user-observable verification, including what some of the desired properties of such a system might be. This will set the stage for Section 3, where we detail open research issues.

2.1 Possible Approaches

Computing Devices that Self-Verify. Self-verification is not an option on today's computing devices, as we cannot trust a potentially compromised device to report its status correctly. Future architectures may change this. One possibility is to include a trustworthy verification subsystem in every device. However, there is still the problem of how to communicate the result of the verification to the user. One could imagine adding secure I/O capabilities to devices to communicate verification results to the user, although such a design choice may increase the cost and complexity of the trustworthy subsystem. Furthermore, if every manufacturer designs their own user-interface for verification, then the user will be confronted with many different interfaces. This could cause confusion, increase the frequency of mistakes, and degrade the users' experience, all of which could result in users disregarding the verification process altogether.

Trustworthy External Device. Instead of a dedicated verification subsystem inside every device, we could build a single verification device. This alleviates the problem of the user having to learn and use several different verification interfaces. In our opinion, this is the most desirable solution from the point of view of ease of use. This dedicated device provides an unambiguous point from which trust originates for the user. We call this device the *iTurtle,* based on analogy with the "Turtles All the Way Down" story, because it is the turtle on which all other turtles stand, i.e., it is the turtle on which all user trust is built. Figure 2 portrays the iTurtle in use.

2.2 Desired Properties

Software Design Simplicity. The software on the iTurtle should be small, since it should be amenable to formal verification or manual audit for security assurance. The software design should avoid the use of cryptographic secrets to eliminate the overhead involved with their maintenance. Such overheads include the use of tamper-resistant or tamper-evident hardware, key management issues like revocation, migration, and regeneration, and vulnerabilities due to lost or stolen iTurtles.

Commodity Hardware. Using commodity hardware will enable inexpensive mass-production of iTurtles. Portions of the software can be borrowed from existing code. Many of the bugs inherent in a new hardware design will have been removed or have known work-arounds. A mature developer community will exist to support iTurtle developers.

Universal Physical Connectivity. The iTurtle should use a ubiquitous physical interface. Probably the best choice today would be USB. A USB-based iTurtle would have the ability to act as a master or slave device, and would be equipped with adapters for the different USB plug sizes.

Wired Interface. The temptation to use wireless interfaces must be avoided, because without physical connectivity, there is no way (without the use of cryptography) for the human to unambiguously identify the device being verified by the iTurtle. Note that using a wired interface still does not address a relay attack where the challenged machine relays an attestation request to another machine [13].

User Interface Simplicity. The interface which tells the user whether or not verification succeeded should be simple enough to be used by untrained novice users. Simplicity of the user interface should also minimize the opportunities for user error. An example of a simple user interface is a dual-color LED capable of showing a red or a green light, as shown in Figure 2.

Small Form Factor. The iTurtle should be small, lightweight, and rugged. This will enable users to always carry the iTurtle with them, e.g., on a keychain.

2.3 Our Proposed Verifier

Based on our desired properties, a USB fob designed with a commodity microprocessor would be a good candidate for the iTurtle. Such hardware already exists and is available inexpensively.

3 Research Issues

Research challenges arise when we try to build a verification system using the iTurtle.

3.1 What to Verify

Attestation mechanisms verify the software configuration of a computing device, raising the question: *How do we define the software configuration of a computing device?* Below we consider two possible answers, and note their difficulties.

Attestation schemes proposed in the literature treat the software configuration of a computing device as all software that has been loaded for execution since the last reboot [2, 6, 12, 14]. However, even with a small set of installed software, the number of possible software configurations can be large. This is because the number of possible configurations is the number of permutations of the subset of loaded software from the set of all installed software. Some of the problems for the iTurtle include: (1) *How does the iTurtle attach meaning to all of these different configurations?* (2) *How does the iTurtle store so many different configurations for many different devices?*

An alternate scheme would be to define a software configuration as the list of all software that is installed on the device. However, this definition does not take into consideration the problems that arise due to interactions between different software components.

We must also consider the question of how to represent the software configuration of a device, i.e., how to assign identities to software. Another question is how to translate a software configuration into a trust decision.

3.2 How to Verify

We now discuss how the iTurtle might verify the software on a computing device, and we identify open research issues. The verification process consists of comparing the software configuration of the device being verified against known-good configurations.

What is a known-good configuration? The user must somehow translate their notion of trust into a set of known-good software configurations. This is a problem with no clear solution. The issue is the semantic gap between a list of program identities and the nebulous notion called trust. The vendor of a computing device might be able to help the users by providing some default known-good configurations for the device.

How does the iTurtle obtain known-good configurations the first time? With trusted computing today, there are at least two possible ways to address this problem. (1) The *Oracle Method* retains a trusted third party as a read-only oracle which provides a list of known-good software configurations to the iTurtle. (2) The *Trusted First Time (TFT) Method* assumes that the system is in a secure state the first time it is verified and compares all subsequent verifications against the first one.

With our verification model, both of these approaches have unresolved issues. Questions which arise with the Oracle Method include: (1) *How does the iTurtle establish an authenticated channel to the oracle?* (2) *How does the user ensure that their privacy is not compromised through oracle queries?*

Question (1) does not arise with TCG-style attestation because the verifier is assumed to be a properly configured general-purpose computing device, whereas the iTurtle is a special-purpose device with limited capabilities. Thus, no sufficiently secure method exists today that would allow the iTurtle to learn known-good configurations. TCG-style attestation tries to address question (2) with techniques such as Privacy CAs [14] or Direct Anonymous Attestation [4]. However, these techniques are too heavy-weight for the iTurtle.

An important question for the TFT Method is: *How does the iTurtle distinguish between a legitimate installation or upgrade and an attack?* The user must somehow convey to the iTurtle that a legitimate upgrade or installation is taking place. Doing this without increasing the complexity of the iTurtle is a challenge.

An additional question for verification is: *How does the user use the same iTurtle to verify multiple devices?* The issue here is one of authenticating the device being verified so that the iTurtle uses the correct known-good configuration during verification. Since the iTurtle does not employ cryptography (recall Section 2.2), we cannot employ cryptographic authentication methods [7, 13]. Also, interoperability demands a standard protocol for communication between the device and the iTurtle. Standardizing such a protocol is an engineering and political challenge.

3.3 What to Do When Verification Fails

Most trusted computing literature does not address procedures for recovering if verification fails, raising the following questions.

How does recovery happen? If the device is compromised, then a trusted entity (the recovery agent) must be involved in recovery. The recovery process may also need to involve the user; however, the level of user expertise required is unclear. It may be necessary to have an expert such as an ISP, a device vendor, or a third party service perform the recovery for a fee. The infrastructure required for involving an expert may be prohibitive.

Where is the recovery agent? The recovery agent can be part of the iTurtle, part of the device, or some combination of both.

If the recovery agent is located on the device, then, for secure recovery, it needs to be isolated from all other software on the device, and the user needs to know that it launched correctly. Technologies such as AMD Presidio [1] and Intel TXT [5] provide only partial solutions in that they only address the isolation issue but do not provide the user with the guarantee of correct launch.

If the recovery is performed completely by the iTurtle, then we could design a *Snapping* iTurtle to help revert ("snap") the system state back to a previously known-good state. However, it is unclear how the iTurtle can obtain sufficient control of the device to perform the snap operation without involving any entity on the device. One approach is to make the iTurtle a bootable device containing a known-good system image.

Performing recovery using a combination of the device and the iTurtle might be a practical approach, but additional work is required to discover the details of which operations need to be performed by each of them.

How is recovery initiated? Is it user-invoked or automatic? Both choices have usability issues. User-invoked recovery gives more control to the user but becomes annoying if failure is frequent. Equipping the iTurtle with dedicated hardware to accept the user's recovery request increases its cost and complexity. Automatic recovery removes control from the user, which could also destroy data, interrupt critical work, etc.

3.4 Trusting the iTurtle

How can the user trust her iTurtle? Since the user cannot directly verify her iTurtle, the only currently available approaches are for the manufacturer of the iTurtle to certify it, or for a trusted third party to certify it, perhaps for a fee. Certification requires standards to be established for the hardware and software components of the iTurtle. If the resulting standard is complex, then certification as well as conformance to standards will be hard. The certification process of TPMs demonstrates this [10]. Thus, we believe there are no satisfactory answers to this question today.

What if the iTurtle is compromised? We would like to avoid the use of tamper-evident or tamper-resistant hardware, therefore, iTurtle compromise is an issue. The challenges here are: (1) *How does the user detect the compromise?* (2) *When the user detects that her iTurtle is compromised, what should she do?*

Periodic inspection and recertification, and fault tolerant design work well for issues that arise due to normal wear and tear, but they are insufficient to address (1) above. For (2), discarding and replacing the iTurtle is not a good answer, because the user will need to reconfigure the new iTurtle to suit her attestation preferences.

4 Conclusion

We have argued for user-observable verification and outlined a possible system and some of the research issues remaining. User-observable verification lends strong privacy properties to trusted computing technologies since the verification process is completely controlled by the user of the computer. Also, users are free to modify and use their computing devices in any way they see fit, even as they do today, while enjoying the security benefits of trusted computing. We hope that user-observable verification will alleviate concerns that trusted computing exists solely for enforcement purposes.

Perhaps in the future the world of attestation will rest on an iTurtle!

Acknowledgements

The authors are grateful to Bryan Parno for his comments and for suggesting the title. The paper also benefited from the insightful feedback of Jesse Walker, Ahren Studer, Jason Franklin, and the anonymous reviewers.

References

[1] Advanced Micro Devices. AMD64 virtualization: Secure virtual machine architecture reference manual. AMD Publication no. 33047 rev. 3.01, May 2005.

[2] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1997.

[3] B. Balacheff, L. Chen, S. Pearson, D. Plaquin, and G. Proudler. *Trusted Computing Platforms – TCPA Technology in Context*. Prentice Hall, 2003.

[4] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2004.

[5] Intel Corporation. Trusted execution technology – preliminary architecture specification and enabling considerations. Document number 31516803, November 2006.

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

[7] K. Goldman, R. Perez, and R. Sailer. Linking remote attestation to secure tunnel endpoints. Technical Report RC23982, IBM, June 2006.

[8] D. Grawrock. *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*. Intel Press, 2006.

[9] C. Rudolph. Covert identity information in direct anonymous attestation (DAA). In *Proceedings of the Workshop IFIP TC11 WG 9.6 / 11.7*, May 2007.

[10] A.-R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy. TCG inside? A note on TPM specification compliance. In *Proceedings of the ACM Workshop on Scalable Trusted Computing (STC)*, 2006.

[11] A.-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In *Proceedings of the 2004 workshop on New Security Paradigms (NSPW)*, 2004.

[12] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of USENIX Security Symposium*, 2004.

[13] F. Stumpf, O. Tafreschi, P. Röder, and C. Eckert. A robust integrity reporting protocol for remote attestation. In *Proceedings of the Workshop on Advances in Trusted Computing (WATC)*, November 2006.

[14] Trusted Computing Group. Trusted platform module main specification, Part 1: Design principles, Part 2: TPM structures, Part 3: Commands. Version 1.2, Revision 94, March 2006.