Catching the Cuckoo: Verifying TPM Proximity Using a Quote Timing Side-Channel (short paper)

Russell A. Fink[1,2], Alan T. Sherman[2], Alexander O. Mitchell[3], and David C. Challener[1]

[1] Johns Hopkins University / Applied Physics Laboratory
[2] University of Maryland Baltimore County / Cyber Defense Lab
[3] Hammond High School, Columbia, MD

Abstract. We present a *Trusted Platform Module (TPM)* application protocol that detects a certain man in the middle attack where an adversary captures and replaces a legitimate computing platform with an imposter that forwards platform authentication challenges to the captive over a high speed data link. This *revised Cuckoo* attack allows the imposter to satisfy a user's query of platform integrity, tricking the user into divulging sensitive information to the imposter. Our protocol uses an ordinary smart card to verify the platform boot integrity through TPM quote requests, and to verify TPM proximity by measuring TPM tickstamp times required to answer the quotes. Quotes not answered in an expected amount of time may indicate the presence of an imposter's data link, revealing the Cuckoo attack. We describe a timing model for the Cuckoo attack, and summarize experimental results that demonstrate the feasibility of using timing to detect the Cuckoo attack over practical levels of adversary link speeds.

Keywords: TPM, Attestation, Timing, Quote

1 Introduction

Despite the proliferation of personal computers and mobile devices, the public relies on kiosk computers for critical security applications such as *Automated Teller Machine (ATM)* banking, filling out tax forms in public assisted offices, entering health insurance information at hospital kiosks, and voting on electronic terminals in large precincts. Kiosk computers are vulnerable to undetected physical attack because they are not controlled as well as personal devices. The *Trusted Computing Group (TCG)* has created the TPM, an embedded cryptographic processor that can convince the user by *attestation* that the correct software was booted on the platform with cryptographic proof of software measurements signed by a private key. A user who knows the corresponding public key can verify the attestation measurements recorded by the TPM, useful for verifying the state of a public kiosk. Attestation, however, requires the user to trust a smart token to issue challenges to the kiosk on his behalf and verify the cryptographic

results. Further, attestation does not detect the Cuckoo attack, defined by Parno in [7] as a corrupt platform forwarding attestation challenges to a machine that can emit the expected responses, leading the user into disclosing sensitive information to the attacker. Therefore, the two major problems with kiosk computing are the user trusting a token to attest the state of the TPM, and the user trusting the physical path between his token and the TPM—in other words, the user cannot confirm the proximity of the TPM that is answering the challenges easily.

In our work to mitigate the Cuckoo attack, we have discovered that a reliable timing side-channel exists with quote command processing through the TPM.[4] We have designed a protocol to exploit this side channel by using an ordinary smart card to issue a sequence of quote requests, measure the amount of time taken to receive the responses, and decide whether the time taken to respond to the quotes is consistent with the expected path components between the smart card and the TPM. In short, we propose using timing to detect the Cuckoo attack. Our contributions are:

- Design and presentation of a TPM quote and timing attestation protocol using an ordinary, inexpensive smart card
- A timing model for the Cuckoo attack and required adversary bandwidth
- Experimental results supporting the use of timing side-channels to verify TPM proximity, revealing the Cuckoo attack.

In this short paper, we define the problem being solved, present a security and timing model, describe our protocol, and give an overview of the experimental results we obtained that demonstrate the feasibility of our approach.

2 Security and Timing Model

The problem we solve begins when the user intends to interact with a specific *kiosk*, but the adversary wants him to interact with an *imposter*. The adversary kidnaps the kiosk and replaces it with the imposter, and forces the kiosk to answer challenges on behalf of the adversary, thereby fooling the user.

This *Man-In-The-Middle (MITM)* attack is described by Parno as a Cuckoo attack where an adversary redirecting attestation challenges to a remote TPM under his control [7]. Our revised model, shown in Figure 1, places the imposter directly in front of the user. The imposter forwards any *Platform Configuration Register (PCR)* challenges to the kidnapped kiosk computer to obtain signed PCR values that the user expects to see. Our revised attack assumes that the user knows the public AIK of the kiosk used to verify quotes, but cannot confirm the physical proximity of the TPM. The attack resists detection by exploiting the gap between the challenger and the verifier.

We can detect Cuckoo attack by timing precisely an exchange of authentication messages between a smart card and a TPM. A notional timing model

[4] A *quote* is a reading of the TPM *Platform Configuration Register (PCR)* values signed by an *Attestation Identity Key (AIK)* belonging to that particular TPM. It is cryptographic evidence that the platform booted the correct software.

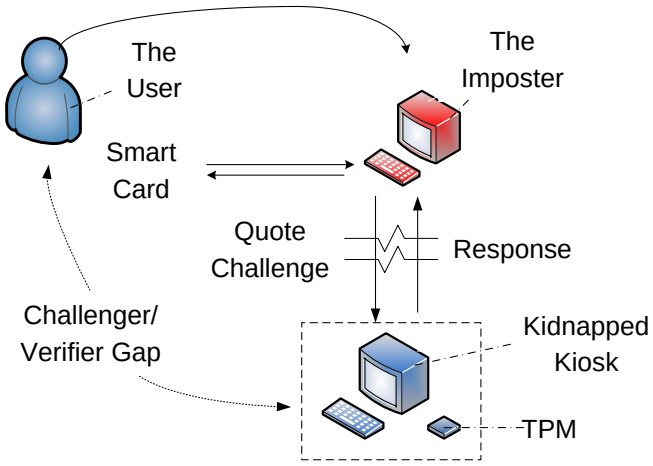

Fig. 1. In the revised Cuckoo attack, the user interacts with an imposter that forwards challenges to the legitimate kiosk, tricking the user into interacting with the imposter. This attack exploits a proximity gap between the challenger (user) and the verifier (kiosk).

for this authentication sequence includes the time for the smart card to issue its challenge, the platform CPU to forward it to the TPM, and the time for the TPM to respond. Repeated trials of this exchange taken during a training phase will reveal some timing noise (measured as statistical variance) caused by network collisions, processor scheduling, card latency, and TPM load. In the Cuckoo attack, the adversary adds additional components to the model, potentially increasing the average transaction time. If we conservatively assume that the extra components in the Cuckoo case contribute no extra noise, then the attack succeeds without detection when the time added by the extra components is no greater than the noise time observed in training. We model the extra components in the Cuckoo attack in Figure 2.

Let the notional timing noise be ϵ, and the Cuckoo attack time be C. If we remove the imposter components from the model, and add back in the notional noise, the non-attack time becomes $C - 2(T'_N + T'_P) + \epsilon$; therefore, the Cuckoo attack cannot be distinguished from expected noise in the non-imposter case when:

$$2(T'_N + T'_P) \leq \epsilon \tag{1}$$

Therefore, this model states that the adversary's connection speed in the hostage case varies linearly with the noise observed during the non-hostage training phase.

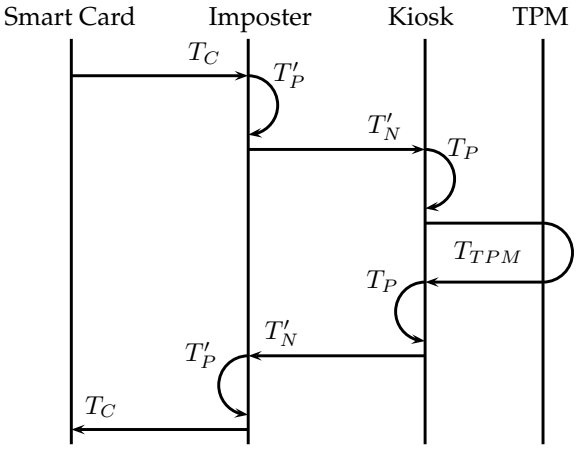

Fig. 2. Timing model for the revised Cuckoo attack. The smart card challenge takes time T_C to arrive at the imposter. The imposter takes takes T'_P to prepare the challenge and T'_N to forward it. The kiosk processes the message in T_P, and the TPM takes time T_{TPM} to respond.

3 Previous and Related Work

Parno [7] first described the Cuckoo attack as a user interacting with a kiosk tampered to forward challenges to the attacker's machine. Parno assumes that the user does not know any public keys of the kiosk TPM, enabling the attacker's (or anyone's) TPM to respond to attestation requests.[5] Our revised Cuckoo attack reverses the placement of the kiosk and the imposter. As with Parno, we assume that the user cannot confirm the physical proximity of the TPM. Unlike Parno, we require that the user knows the public AIK of the kiosk.

The Cuckoo attack is also called a Masquerade attack. Stumpf *et al.* proposed a TPM solution in [9] that establishes an encrypted session tunnel following attestation using a shared key exchanged during attestation, preventing interactions with the imposter. Goldman *et al.* proposed a similar system in [4]. While these are feasible when both communicating endpoints are computers, they do not fit the user-kiosk model where the human interacts directly with the host.

The TCG specifies the TPM, an embedded cryptographic processor and non-volatile storage device that can generate keys and use them securely according to policy. Our protocol uses the TPM to verify the correctness of platform identity and software state, and to keep accurate time for the smart card. The TPM is explained in specifications [10], and the software programming interface is

[5] While we disagree that distributing public portions of AIKs is difficult, we concede that *Public Key Infrastructure (PKI)* is often implemented incorrectly in practice.

explained in [11]. Challener *et al.* covers TPM application programming in [1]. Fink *et al.* give an example of TPMs enforcing election security policy using platform state in [2].

Attestation involving cryptography is not practical for a human without assistance. Trusted computing supports attestation by using TPM to engage in a special challenge-response protocol with networked computers, called *Trusted Network Connect (TNC)*, described in [12]. However, TNC requires back-end infrastructure to verify challenges. We use a smart card in place of the back-end to let the user verify the platform.

We use timing side channels to verify TPM proximity to our smart card. Seshadri *et al.* created SWATT [8] that attests embedded software using timing. Gardner *et al.* uses memory access latency to attest software running on general purpose computers in [3]. These systems show the feasibility of using timing for attestation.

4 Protocol

Our authentication and proximity verification protocol uses a smart card to issue a fixed number of TPM_QUOTE requests to the local platform, measuring the time taken for these calls to complete as a group. The smart card verifies the quotes, and compares the aggregate response time with some clean room training set of timings and noise established previously. If the quote responses are correct and the response time is within the noise margin of the training times, the smart card releases a passphrase to the local platform indicating attestation success indicating the attestation decision of the smart card to the user. If either the timing or quote verification fails, the smart card releases a duress passphrase.

We assume that the adversary has a network of finite speed to connect his imposter to the kiosk. We assume that the adversary does not know the passphrases, nonces, or the AIK private key. We also assume that the TPM is working correctly. Figure 3 pressents the protocol in more detail.

5 Experiments

We conducted experiments to verify our approach. The experiments timed the interactions between the operating system and TPM separately from the interactions between the smart card and operating system—this simplified setup allowed quick validation in lieu of a full end-to-end experiment. We calculated the average values and variances of the challenge/reponse times in each experiment, then combined the variances to compute the critical value of the one-tailed t distribution that dictates the required speed of the adversary's network.

5.1 OS to TPM

We created the experiment using a STmicroelectronics TPM running on a Dell OptiPlex 755. We developed two software programs for Linux, one using IBM's

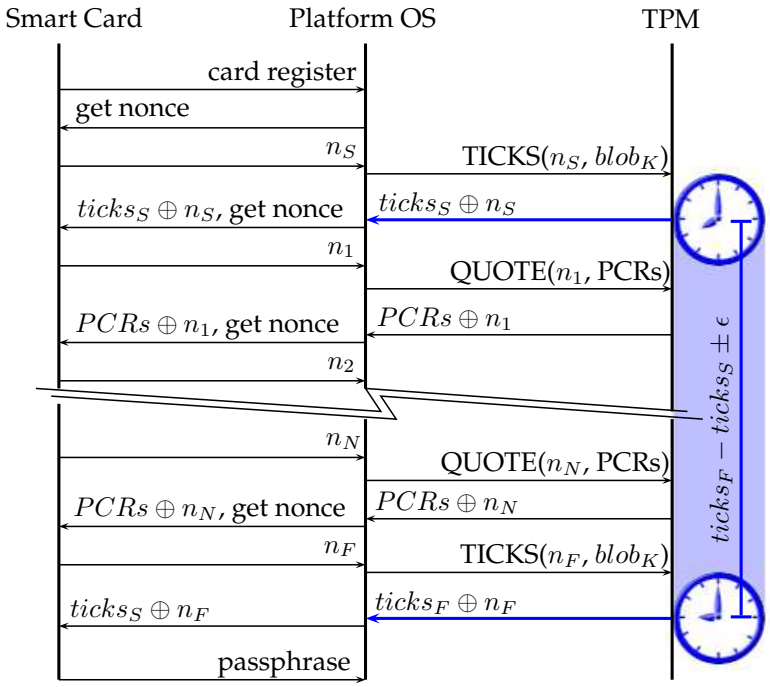

Fig. 3. A smart card issues a challenge nonce n_S to the TPM for an initial tickstamp, $ticks_S$, recording the start of the timed quote loop. The smart card issues unique nonces to obtain quotes. After receiving a final tickstamp, $ticks_F$, the smart card verifies the responses and loop time, then releases a positive or duress passphrase. The expected loop time is measured in an initial training phase. [$\oplus$ signifies cryptographic binding with the TPM's AIK.]

Count	OS Time	Tickstamps
1	148.792325	149.934
351	148.800337	149.945
	148.800325	149.944
	148.800326	149.945
	. . .	. . .
45	148.804325	149.949
	148.804315	149.946
	148.804325	149.947
	. . .	. . .
1	148.808326	149.952

Count	OS Time
799	1.3997048
	1.3997401
	. . .
1	1.4006945
790	1.9997026
	1.9997082
	. . .
8	2.0000521
	2.0009368
	. . .

Table 1. Sample timing data, showing results for TPM (left) and smart cards (right). Times are of repeated challenge/response runs, expressed in seconds. OS (wall clock) time is compared with tickstamp time. Multi-modal harmonics (counts per group shown) differed by 0.004 secs for TPM and by 0.6 secs for smart cards.

software TPM emulator and accompanying command-line utilities [6] and the other built on TrouSerS [5]. We collected data for runs that consisted of 200 challenge/response messages, and did several hundred runs to form the population. We terminated many extraneous operating system services during the runs.

A summary of the TrouSerS data is presented in Table 1. The multi-modal clustering resembles *harmonics* that occur at intervals of 0.004 seconds; these are due likely to device interrupts and process scheduling of our test program by the Linux operating system.

5.2 OS to Smart Card

We timed different *Subscriber Identity Module (SIM)* Java cards running under Windows XP, and implemented a simple data collector using C#—we have approximated the timing variance of reading data from a program running on the card with that of reading a simple data value from the card. We timed 1,600 samples of 200 message transactions per sample. Data are shown in Table 1.

5.3 Analysis

Using the data from the largest harmonic groups, we computed the population standard deviation and used it to determine the one-tailed t test critical value. At the 95% confidence limit, our experiments a SIM variance of 3.2739×10^{-10} seconds and a TPM variance of 2.4925×10^{-7} seconds. The combined σ is 4.9958×10^{-4}, giving a critical value of 8.1931×10^{-4}. The attestation challenges consume 2,806 bytes per loop, for a total of 1.5 megabytes after adding in nominal packet headers. This requires the adversary to have a minimum network speed of 14.8 gigabits per second to avoid detection by our protocol.

6 Results and Conclusions

The predicted gigabit speed is faster than modern wireless technologies, meaning that the adversary must use wired, and possibly fiber, connections to carry out the Cuckoo attack. Such wired connections are both visible and expensive, and are revealed by simple physical inspection of the kiosk.

In conclusion, we have developed a protocol that works with an ordinary smart card and a TPMs to verify identity, state, and physical proximity of the platform. The protocol uses inexpensive technologies and enables practical proximity attestation for kiosk-style public computers. The attestation smart card is simple enough to be validated independently, *e.g.* in Seshadri [8].

7 Future Work

We must develop a full path, smart card to TPM experiment, and investigate environmental factors. We must characterize the probability of Type 1 errors.[6]

[6] The timing harmonics are evidence that Type 1 errors are likely.

A variety of smart cards should be tested, including identical models of the same cards. We should determine the optimal training time and number of loops to minimize the time needed for attestation.

Acknowledgments

We thank Ryan Gardner for formative discussions, and also T. Llanso, F. Deal, B. Benjamin, E. Reilly, and members of the UMBC *UMBC Cyber Defense Lab (CDL)* for good suggestions. We thank April Lerner, gifted and talented coordinator for Hammond High School, for lending us a promising new researcher.

References

1. D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn. *A practical guide to trusted computing*. IBM press, Upper Saddle River, NJ, 2007. ISBN 978-0132398428.
2. Russell A. Fink, Alan T. Sherman, and Richard Carback. TPM meets DRE: Reducing the trust base for electronic voting using trusted platform modules. *IEEE Transactions on Security and Forensics*, 4(4):628–637, 2009.
3. Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Detecting code alteration by creating a temporary memory bottleneck. *IEEE Transactions on Security and Forensics*, 4(4), 2009.
4. K. Goldman, R. Perez, and R. Sailer. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the first ACM workshop on Scalable trusted computing*, pages 21–24. ACM, 2006.
5. IBM Corporation. The Trusted Computing Software Stack (TrouSerS) software library. Available at `http://sourceforge.net/projects/trousers/`, 2008. Last accessed Feb 3, 2011.
6. IBM Corporation. Software TPM emulator. Available at `http://ibmswtpm.sourceforge.net/`, 2010. Last accessed June 23, 2010.
7. B. Parno. Bootstrapping trust in a trusted platform. In *Proceedings of the 3rd conference on Hot topics in security*, pages 1–6. USENIX Association, 2008.
8. A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: SoftWare-based ATTestation for embedded devices. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 272–282. IEEE, 2004.
9. F. Stumpf, O. Tafreschi, P. Röder, and C. Eckert. A robust integrity reporting protocol for remote attestation. In *Second Workshop on Advances in Trusted Computing (WATCâ06 Fall)*. Citeseer, 2006.
10. Trusted Computing Group. TCG TPM specification version 1.2, revision 103. Available at `https://www.trustedcomputinggroup.org/specs/TPM`, 2008. Last accessed on Mar 15, 2008.
11. Trusted Computing Group. The TCG Software Stack. Available at `http://www.trustedcomputinggroup.org/developers/software_stack`, 2009. Last accessed Sep 1, 2009.
12. Trusted Computing Group. The TCG Trusted Network Connect. Available at `http://www.trustedcomputinggroup.org/developers/trusted_network_connect/`, 2009. Last accessed Sep 1, 2009.