Computer Communications 131 (2018) 35–38

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Initializing trust in smart devices via presence attestation

Xuhua Ding[a], Gene Tsudik*,[b]

[a] Singapore Management University, Singapore
[b] University of California, Irvine, USA

ABSTRACT

Many personal computing and more specialized (e.g., high-end IoT) devices are now equipped with sophisticated processors that only a few years ago were present only on high-end desktops and servers. Such processors often include an important hardware security component in the form of a DRTM (Dynamic Root of Trust for Measurement) which initiates trust and resists software (and even some physical) attacks. However, despite substantial prior research on trust establishment with secure hardware, DRTM security was always considered without any involvement of the human user, who represents a vital missing link. This prompts an important challenge: *how can a user (owner) determine whether a genuine DRTM is currently active on his or her device?* We believe that, in order to address this challenge, a new security service – called "Presence Attestation" ($\mathcal{PA}$) – is needed. While by itself, has only ephemeral value, it can be used to set up a long-term secure channel between the device's DRTM and another device with the user's trust. In this paper, we outline the notion of which is based on mandatory (though, ideally minimal) user participation, overview recent results, and discuss directions for future research.

1. Introduction

Many current computing device processors are equipped with a dedicated hardware security feature, called DRTM: *Dynamic Root of Trust for Measurement*. Typically, DRTM is itself a part of a larger and more general-purpose security component that provides other features and some degree of tamper-resistance, e.g., Intel TXT [8], Interl SGX, AMD SVM [1], and ARM TrustZone [2]. Devices with such processors include laptops, tablets and smartphones, as well as high-end IoT devices. A typical DRTM is designed to withstand kernel-level attacks or even some physical attacks. When activated at run-time, it securely measures and launches software, which may itself further measure and load another layer of software. Multiple iterations of measure-then-launch operations form a trust chain rooted in DRTM, which allows a remote entity (verifier) to establish trust in the DRTM-equipped device, after checking integrity of the latter's software stack.

Popularity of DRTM motivated some new directions in system security research. Several designs [4,5,10,15,17] have been proposed to cope with kernel-level threats by using various DRTM instantiations to ensure security of the Trusted Computing Base (TCB).

1.1. User involvement

However, most prior efforts either overlooked or side-stepped an important factor – **the human user**. As noted by Parno et al. [14], it is challenging for a user to bootstrap trust in DRTM, since she is not assured that the chain of trust is indeed rooted in **her** DRTM. This process of bootstrapping trust is referred to as *Presence Attestation* ($\mathcal{PA}$). The main reason why $\mathcal{PA}$ is challenging is the difficulty for a user to authenticate her own DRTM – a component that is not externalized, i.e., hidden from user's view, and lacks any direct user interface. Consequently, although DRTM is trusted in general, the user can not determine whether her specific DRTM is active and engaged. In particular, malware (including a compromised OS) can impersonate the user's DRTM using the so-called *cuckoo attack* [14]. This attack type, shown in Fig. 1, assumes that the user's device has malware, and the adversary controls an accomplice device of the same type, which has its own DRTM. This DRTM is *not* compromised, since doing so would require much more adversarial effort, i.e., a physically invasive attack that violates tamper-resistance features.

A stronger variant of the basic cuckoo attack is the *analog cuckoo attack*, whereby the adversary deploys additional analog devices (e.g., a display and/or a speaker) near the accomplice device, in order to mimic the physical environment of the victim. We beliee that cuckoo attacks are realistic, mainly because they do not require any physical presence by the adversary, i.e., such attacks can be fully automated, with collaboration among malware on the user's device and the accomplice device.

Typically, modern DRTM architectures assign to each DRTM instance a unique ID, usually associated with, or derived from, a (also unique) public key. Each DRTM keeps the corresponding private key in its secure storage. This provides the means to distinctly identify and authenticate an individual DRTM, e.g., a DRTM could trivially sign a challenge from anyone, including its user. Unfortunately, the crux of

* Corresponding author.
E-mail addresses: xhding@smu.edu.sg (X. Ding), gts@ics.uci.edu (G. Tsudik).

https://doi.org/10.1016/j.comcom.2018.07.004
Received 21 May 2018; Received in revised form 26 June 2018; Accepted 1 July 2018
Available online 20 July 2018

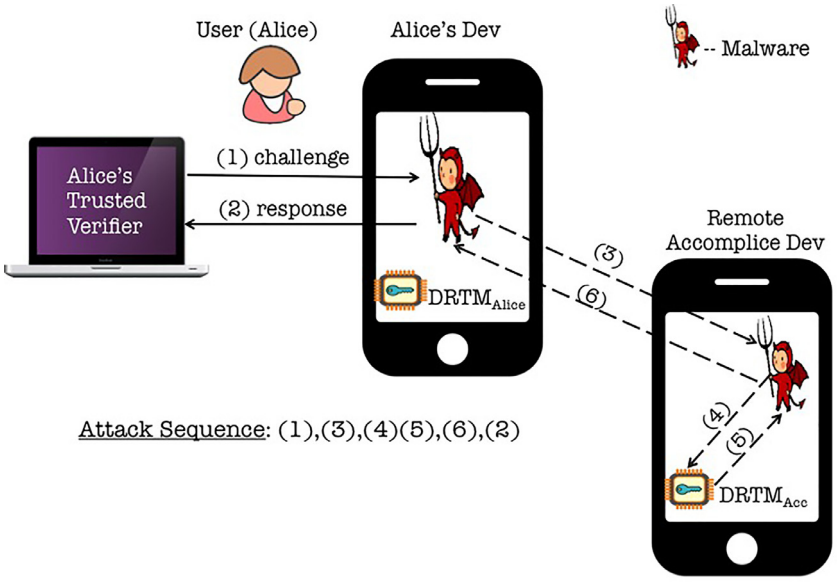

Fig. 1. Cuckoo attack example [14]. DRTM$_{Alice}$ in Alice's Dev is not active. Malware residing in it uses DRTM$_{Acc}$ of the accomplice device to produce a legitimate response to the verifier which fails to determine the origin of the response.

the problem is that there is no easy way for the human user to authenticate her DRTM using public key cryptography.

1.2. Why bother with $\mathcal{PA}$?

It is natural to wonder why the user should care about establishing trust in her own device's DRTM and resisting cuckoo attacks. For instance, the user hopes to ensure that the DRTM on her phone is currently in action when performing an Over-The-Air (OTA) firmware update. Beyond gaining an immediate (and very temporary) feeling of security, there are few, if any, practical reasons for the user to verify that she is "talking" to the correct DRTM. However, the main reason for $\mathcal{PA}$ is that it – with the user's aid – facilitates establishment of a long-term authenticated and secure channel between the user device (denoted as Dev hereafter) and another entity already trusted by the same user. This is a very important step, since having a secure channel serves as the stepping-stone for other security services, such as secure reset, secure software/firmware update, and more general secure computation.

1.3. $\mathcal{PA}$ vs. device pairing

At the first glance, it might seem that $\mathcal{PA}$ is the same as, or very similar to, the well-known and thoroughly explored topic of Secure Device Pairing (SDP) [6,13]. Basically, SDP addresses the setting where the user needs to securely pair two (usually wireless) devices that have no prior knowledge of each other's credentials or identities. (Note that "secure pairing" in SDP context is equivalent to establishment of a secure channel in ours.) Key similarities between SDP and $\mathcal{PA}$ are:

- Goal of setting up a long-terms secure channel (association) between two previously unfamiliar devices.
- Unavoidable user involvement, and need to minimize user burden.
- Human-imperceptible communication between devices, including users' inability to detect spurious or extraneous communication.
- Threats of various attacks during the process.

Despite these common features, there are some important differences:

- SDP assumes both devices are trusted by the user, while $\mathcal{PA}$ assumes that only the verifier is trusted.
- SDP assumes that devices do not have a common PKI, while in $\mathcal{PA}$ they might.
- SDP assumes no malware presence on either device, while $\mathcal{PA}$ allows Dev to have malware.
- SDP assumes no hardware security features, while $\mathcal{PA}$ requires Dev to have an inviolate DRTM.
- The main threat in SDP is due to so-called *evil twin* attacks, wherein, a nearby adversary (in the form of any device) impersonates one of the devices being paired. In contrast, the main threat in $\mathcal{PA}$ is due to *cuckoo attacks*, wherein, an arbitrarily remote accomplice device (of the same type as Dev) works in concert with malware resident on Dev to subvert the protocol.

$\mathcal{PA}$ is a more difficult problem than device paring, as it considers a stronger adversarial model. A $\mathcal{PA}$ scheme can be used as a building block to construct a device paring scheme. To pair two devices, we can instantiate two sessions of $\mathcal{PA}$ for each of them. As a result, both devices would share a key with the verifier device, which can then act as a middleman to set up a secure connection between them.

2. Current approaches

Two general mitigation directions for cuckoo attacks were proposed and discussed in [14]. The first relies on a hardware-based secure channel, e.g., a special-purpose I/O interface through which an external verification device directly interacts with DRTM. The second establishes a cryptographically secured communication channel and requires the user to have prior (or obtained in real time) knowledge of the public key of her specific DRTM.

The former offers stronger security and better usability, i.e., lower user burden. For example, a smartphone with a dedicated LED light that is securely hard-wired to TrustZone can confirm (e.g., when blinking) to the user that her DRTM is now active. The same blinking light can be used as an analog channel between the new device and the user's already trusted component, e.g., by using the camera to capture blinking patterns.[1] Alternatively, a special-purpose micro-USB port (also securely hard-wired to the user device's DRTM) would allow the user to

[1] Other similar methods are possible, e.g., via NFC.

string a cable between two devices and be assured that her device's DRTM is active. However, few current DRTM-equipped devices has such features and implementing them would require cooperation of device manufacturers. Furthermore, all such techniques require dedicating a human-perceptible interface (e.g., a wired micro-USB port or LED light) exclusively to security functionality of the device. Therefore, even if chip-makers and device manufacturers were to collaborate, it is unclear whether the latter would want to do this.

The latter approach also requires manufacturers' cooperation – though to a lesser extent – in order to convey the DRTM's public key identifier to the user in an out-of-band fashion, e.g., by etching it on the device exterior, printing it on the original packaging or delivering it by post or secure email. In any case, dealing with the DRTM's public key represents an additional burden for the user. Furthermore, while this approach might be viable for new devices, it is unsuitable for those that are re-purposed or re-sold.[2]

Another simple and intuitive cuckoo attack mitigation approach is to make sure that the user's device can not communicate to any accomplices. In theory, this can be achieved by muffling all communication interfaces, which, on modern smartphone-class devices usually include at least: Bluetooth, NFC, WiFi, Cellular, and maybe even speaker and microphone.[3] Besides posing a moderate burden for the user, resident malware can simply pretend to shut down all these interfaces while not actually doing so. The user can not verify either way since radio communication is obviously imperceptible to humans. A somewhat more drastic approach is to enclose the device in a Faraday cage or a similar environment that inhibits communication with the outside. This approach has been explored in [9]. It is not cheap, needs extra equipment and is unrealistic for a general user setting. Moreover, it has been discovered recently that even a Faraday cage or an air gap does not protect against outside leakage [7].

Another important aspect is that the user's trust in her device should not be based solely upon physical availability of a DRTM. Instead, it should be based on availability and security of *software* directly measured and loaded by the DRTM, which constitutes the TCB of the device. We refer to this initial software in the trust chain (immediately following DRTM) as the *trust anchor* or $\mathcal{TA}$. Note that, if a DRTM can not ensure runtime security of the $\mathcal{TA}$, its measurements are of little value since they only momentarily reflect static software integrity[4].

As part of more recent work, Zhang et al. [16] suggested dividing the $\mathcal{PA}$ procedure into two consecutive phases:

Existence Checking The user (assisted by her already trusted verifier device) checks whether a genuine DRTM has launched the $\mathcal{TA}$ and is currently interacting with her. The main goal here is to ascertain whether **any** DRTM is involved in the on-going interaction, not necessarily the one on the user's device.

Residence Checking The user verifies whether the $\mathcal{TA}$ engaged in the preceding step actually resides on her own device.

One important aspect of this division is that the second step is cryptographically bound to the first by an ephemeral secret key agreed upon by the trusted verifier and the DRTM. Also, the implicit logical link between the two steps is based on $\mathcal{TA}$'s runtime security being ensured by the DRTM. Indeed, this is a widely adopted trust assumption in many DRTM-based secure systems.

The first phase is fairly straight-forward and burden-free for the user, since it merely involves the two devices (verifier and Dev) agreeing on a fresh ephemeral secret key. However, the second phase is more challenging, since it requires a joint effort by the user and the verifier to make sure that the DRTM from the first phase resides on Dev.

Based on the general approach outlined above, Zhang et al. [16] presented three concrete $\mathcal{PA}$ schemes that differ in the second phase. By taking advantage of hardware DRTM's security assurance and the $\mathcal{TA}$'s software capability, these schemes allow a user to establish trust in Dev after confirming that DRTM and the $\mathcal{TA}$ are active there. Proposed schemes are based on different physical properties: location, scene and sight. The sight-based scheme achieves strongest security even against the analog cuckoo attack, while the other two offer better usability commensurate with slightly weaker security. We summarize them below and refer to [16] for details.

The location-based scheme assumes that the accomplice device is not in (or near) the same location as the user's device. Under DRTM's protection, the $\mathcal{TA}$ securely measures the current GPS location of its hosting device and reports it to the verifier via the authenticated channel established in the first phase. The user then determines the residence of the active $\mathcal{TA}$ and DRTM according to the location conveyed by the verifier. Note that, the $\mathcal{TA}$ must acquire Dev's actual true location, i.e., if the adversary is capable of GPS spoofing, this scheme is insecure. Furthermore, though GPS is ubiquitous on smartphones and other general personal devices (such as laptops, tablets), it might not be available on all DRTM-equipped devices, e.g., smart appliances. Also, GPS can lack precision, and GPS signal might be unavailable in indoor settings.

The scene-based scheme assumes that the accomplice device can not observe the user's immediate environment. It is essentially a challenge-response protocol, whereby the user randomly picks an object (or a set thereof) in her vicinity as the challenge fed to Dev via the camera. The $\mathcal{TA}$ securely accesses the camera buffer and reports the captured raw image as the response to the verifier via the authenticated channel. The latter verifies the response and displays the image (scene). The user checks whether it matches the chosen object(s). The security prerequisite is that the $\mathcal{TA}$ must acquire the real image captured by local hardware, i.e., not by malware. The scheme is not secure against the analog cuckoo attack. Malware on Dev can forward the captured image to the colluding monitor which immediately displays it to the $\mathcal{TA}$ on the accomplice device. Hence, the adversary can still return a correct response.

Finally, the sight-based scheme is essentially a challenge-response protocol over the analog light-of-sight channel using the verifier's screen to send data (by displaying a barcode) and Dev's camera to receive (by taking a photo). This approach leverages the unavoidable latency of the analog line-of-sight channel. Considering an analog cuckoo attack setting, if malware (instead of $\mathcal{TA}$) on Dev takes a photo of the verifier's screen, it needs to send this photo to an accomplice device to display in order to impersonate the verifier. Another accomplice device with the same type of DRTM can take its own photo (with the help of its $\mathcal{TA}$) and then signal the verifier over the previously established secure channel. Thus, a cuckoo attack involves at least one extra analog I/O operation. Since the extra photo capture (an analog operation) incurs tens of milliseconds of additional latency, the attack can be detected by the verifier.

3. Open problems & future directions

The three schemes overviewed above represent the first real step in cuckoo attack-resistant $\mathcal{PA}$. Nonetheless, further research is needed to design better (i.e., more usable, secure and versatile) $\mathcal{PA}$ schemes. One possibility is to use so-called *distance bounding* (DB), a suite of protocols that allow one entity to upper-bound physical distance to another [3]. A DB protocol could be used in the second phase of $\mathcal{PA}$ (i.e., residence-checking) to determine maximum distance from the verifier to Dev. This would involve almost no user burden other than bringing Dev reasonably close to the verifier and then checking whether the upper-

[2] This is because the previous owner, being a potential adversary, can preconfigure the device with malware.

[3] We list only wireless interfaces, since wired ones, such as USB, can be easily muffled.

[4] For the same reason DRTM is not used in the literature to directly launch the OS, which is itself vulnerable to runtime attacks.

bounded distance determined via DB is correct. However, DB protocols require very precise (nanosecond-level) clocks, which are not yet a realistic feature of most smart devices. Another possibility is to leverage contextual security schemes [11,12] that rely on contextual information as the basis for access control or authentication. However, $\mathcal{PA}$ considers the kernel-privileged adversary. Hence, it is more challenging to securely sense the ambient environment without relying on the OS. We now summarize several other potential directions for future work.

Batch $\mathcal{PA}$. Current $\mathcal{PA}$ schemes only consider establishment of trust in a single device. Sequentially applying such schemes to verify DRTM presence in multiple devices (e.g., in a populous office setting or smart home full of various gadgets) is unscalable and inefficient. In these settings, some type of *batch* $\mathcal{PA}$ scheme is needed to (ideally) concurrently establish trust in multiple devices. For example, a common challenge can be broadcast by the verifier to all participating devices, and the verifier can use a batch verification algorithm to validate all responses. The main difficulty in designing batch $\mathcal{PA}$ is how to choose a suitable challenge that suits heterogenous devices with different sensing capabilities.

Trust propagation. The original motivation of $\mathcal{PA}$ is to bootstrap the user's trust in her device. Hence, it is reasonable to rely on a previously trusted and (assumed) secure device as the verifier. A successful $\mathcal{PA}$ instance implies that the user's trust is propagated from the verifier to the new device. It is interesting to consider how to leverage this newly acquired trust. One appealing direction is to use the new device as the extension of the verifier to establish trust in other devices. Consider an example scenario where a system administrator has a smartphone (freshly attested via $\mathcal{PA}$) that communicates with the verifier over a secure channel to attest WiFi access points in the office. As a result, trust propagates from the smartphone to other devices. The rich sensing capability of the smartphone could greatly simplify trust propagation. Without requiring the user to check correctness of a physical property measurement, the smartphone can compare the report from the attesting device with its own measurement. This expands the scope of suitable physical properties. For example, while a user can not measure or compare measured wireless signal strength (RSSI), the smartphone can easily do so.

Usability Assessment. $\mathcal{PA}$ schemes, similar to secure device pairing techniques, involve varying degrees of user participation and therefore burden. Even an ostensibly simple task (on paper) can be confusing, unnatural or unpleasant to an average device user who is likely not technology-savvy. Although user tasks are easy enough to describe and compare conceptually, it is impossible to assess their usability and acceptability without experimenting with actual users. To this end, extensive user studies are necessary.

4. Conclusions

In summary, the threat of cuckoo attacks coupled with lack of a secure DRTM-user interface makes it very challenging to determine residence of a specific DRTM. This, in turn, limits the benefits of DRTM's strong security guarantees. As a countermeasure to cuckoo attacks, presence attestation securely verifies that the DRTM of a potentially compromised device (physically controlled by a user) is active. A successful presence attestation outcome bootstraps the user's trust on her device, as well as facilitates initial establishment of an authenticated channel between devices. Although several schemes have been proposed, striking a good balance between security, usability and versatility remains to be a challenge. Other unaddressed issues include: how to verify DRTM presence in a batch fashion, how to extend trust from one device to others, as well as how to evaluate and compare usability of various presence attestation schemes.

Acknowledgements

The second author was supported in part by: (1) DHS, under subcontract from HRL Laboratories, (2) ARO under contract W911NF-16-1-0536, and (3) NSF WiFiUS Program Award 1702911.

References

[1] AMD, Secure virtual machine architecture reference manual, Technical Report, Advanced Micro Devices, 2005.
[2] ARM, ARM security technology - building a secure system using trustzone technology, (http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf).
[3] G. Avoine, M.A. Bingöl, I. Boureanu, S. Čapkun, G. Hancke, S. Kardaş, C.H. Kim, C. Lauradoux, B. Martin, J. Munilla, A. Peinado, K. Rasmussen, D. Singelée, A. Tchamkerten, R. Trujillo-Rasua, S. Vaudenay, Security of distance bounding: a survey, ACM Comput. Surv. 4 (2017).
[4] A.M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, W. Shen, Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world, Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS), (2014).
[5] A.M. Azab, K. Swidowski, R. Bhutkar, J. Ma, W. Shen, R. Wang, P. Ning, SKEE: a lightweight secure kernel-level execution environment for ARM, Proceedings of NDSS, (2016).
[6] M. Fomichev, F. Álvarez, D. Steinmetzer, P. Gardner-Stephen, M. Hollick, Survey and systematization of secure device pairing, IEEE Commun. Surv. Tutorials 20 (1) (2018).
[7] M. Guri, Beatcoin: Leaking private keys from air-gapped cryptocurrency wallets, 2018, (https://arxiv.org/pdf/1804.08714.pdf).
[8] Intel Corporation, Intel trusted execution technology (Intel TXT) software development guide, 2009.
[9] C. Kuo, M. Luk, R. Negi, A. Perrig, Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes, Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys), (2007).
[10] J.M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, A. Perrig, Trustvisor: efficient TCB reduction and attestation, Proceedings of the 2010 IEEE Symposium on Security and Privacy (S&P), (2010).
[11] M. Miettinen, N. Asokan, F. Koushanfar, T.D. Nguyen, J. Rios, A.-R. Sadeghi, M. Sobhani, S. Yellapantula, Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIACCS, (2015).
[12] M. Miettinen, S. Heuser, W. Kronz, N. Asokan, A.-R. Sadeghi.
[13] S. Mirzadeh, H. Cruickshank, R. Tafazolli, Secure device pairing: a survey, IEEE Commun. Surv. Tutorials 16 (1) (2014).
[14] B. Parno, J.M. McCune, A. Perrig, Bootstrapping Trust in Modern Computers, Springer, 2011.
[15] A. Vasudevan, S. Chaki, L. Jia, J. McCune, J. Newsome, A. Datta, Design, implementation and verification of an extensible and modular hypervisor framework, Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P), (2014).
[16] Z. Zhang, X. Ding, G. Tsudik, J. Cui, Z. Li, Presence a estation: the missing link in dynamic trust bootstrapping, Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS), (2017).
[17] Z. Zhou, V.D. Gligor, J. Newsome, J.M. McCune, Building verifiable trusted path on commodity x86 computers, Proceedings of the 33rd IEEE Symposium on Security and Privacy, S&P, (May 2012).