JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Michael Preisach, BSc
1155264

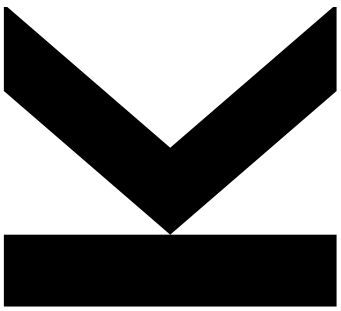
Submitted at
Institute for Networks and Security

Supervisor and First Examiner
Univ.-Prof. DI Dr.
René Mayrhofer

Second Examiner
DI **Tobias Höller**

August 2, 2020

Project Digidow: Biometric Sensor



Master Thesis

to obtain the academic degree of

Master of Science

in the Master's Program

Computer Science

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

JOHANNES KEPLER
UNIVERSITY LINZ

Abstract

What is it all about? Why is that interesting? What is new in this thesis? Where is the solution directing to?

Zusammenfassung

Das am Institut für Netzwerke und Sicherheit entwickelte Projekt *Digital Shadow* benötigt in vielen Bereichen ein prüfbares Vertrauen um eine Erkennung von Nutzern anhand ihrer biometrischen Daten zu erkennen und Berechtigungen zuzuteilen. Das Vertrauen soll dem Nutzer die Möglichkeit geben, die Korrektheit des Systems schnell und einfach zu prüfen, bevor er/sie disesm System biometrische Daten zur Verfügung stellt Diese Masterarbeit beschäftigt sich nun mit den existierenden Werkzeugen, die ein solches Vertrauen schaffen können. Das implementierte System kombiniert diese Werkzeuge, um damit sensible Daten von Nutzern aufzunehmen und im Netzwerk von Digital Shadow zu identifizieren. Es soll dabei sicher gestellt sein, dass eine fälschliche Verwendung der sensiblen Nutzerdaten ausgeschlossen wird. Anhand dieses Systems werden die Eigenschaften einer vertrauenswürdigen Umgebung für Software diskutiert und notwendige Rahmenbedingungen erläutert.

JOHANNES KEPLER
UNIVERSITY LINZ

Contents

1 Introduction

We all live in a world full of digital systems. They appear as PCs, notebooks, cellular phones or embedded devices. Especially the footprint of embedded computers became so small that they can be used in almost all elctrical devices. This product category form the so called *smart* devices.

With all these new devices a lot of societal problems could be solved in the past few decades. Many of them automate services to the public like managing the bank account, public transportation or health services. The list of digital service is endless and will still grow in the next decades.

The downside of all these digital services is that using these services generate a lot of data. Besides of the intended exchange of information, many of the services try to extract metadata as well. Metadata answers some of the following questions. Which IP is connected? What kind of device is that? Is the software of the device up to date? Was this device here in the past? What else did the owner on the device? This list of questions can be continued arbitrarily. Reselling the metadata brings the product manufacturer more margin on the product and hence more profit. Consequently the market for metadata is growing with the Internet itself. The result is a loss of trust in all kind of connected devices and software. A User cannot know what is happening on a device she is using.

The Institute for Networks and Security therefore introduced the project DigiDow. It introduces a digital authentication system, which minimizes the generation of metadata and hence preserves privacy for all users of the system.

1.1 Project DigiDow

The Project *Digital Shadow* is under ongoing development at the Institute of Networks and Security and creates a scalable system for authentication. Key feature is privacy by design and a provable system to create trust to the end user.

At this early stage the interfaces and interaction points are not fully defined.

This is a brief description of the process of authentication:

1.2 Biometric Sensor use case in DigiDow

derive the use case of the Biometric sensor out of the above model.

1.3 Goals and Definitions

You should be able to attach a variety of sensors to the system. The system should then fulfill the following requirements

- *Sensor Monitoring.* The System should be able to monitor the sensor itself.
- *System Monitoring.* It should be possible to track the state of the system. Especially every modification of the system should be detected.
- *Freshness of Sensor Data.* To prevent replay attacks, the system should proof that the provided biometric data is captured live.
- *Integrity of Sensor Data.* As it is possible for an adversary to modify the provided data during the capturing process, integrity should guarantee that the data originates from the BS.
- *Confidentiality of Sensor Data.* It should not be possible to eavesdrop any sensitive data out of the system. Furthermore almost all kinds of metadata (e. g. information about the system or network information) should not be published
- *Anonymity.* Given a message from a BS, an adversary should not be able to detect which BS created it

- *Unforgeability.* Only honest BS should be able to be part of the DigiDow network. Corrupt systems should not be able to send valid messages.

Usage Model of Biometric Sensor

This thesis will describe a system, which is part of the Digital Shadow network. Therefore it has to meet the common principles in information security, namely:

- *Availability*:
- *Integrity*: ISO 27000 (Data Integrity)
- *Confidentiality*: ISO 27000

Upon AIC it should be possible for users to prove honesty of the system. This is what *trust* defines in information security

1.3.1 Requirements

- given a set of software, this system should provide information that exactly this version of software is running on the system. (Integrity)
- The system must furthermore show that it is a member of valid biometric sensors (Attestation)
- All the given information should be anonymized. It should not be possible to gain any other information about the system (Anonymity)
- It should be ensured that no sensitive data is stored at the biometric sensor

Scope of this thesis is on implementing the system from from hardware to application layer. Is is not supposed to think about the network communication.

1.4 Description of structure

1. What exists out there?
2. What is the theoretical solution
3. What about the implementations used - what is the limitation of the used tools?
4. How far are we? what has to be considered next?

2 Related Work

There exist already many interesting projects and implementations which touch the field of trusted computing. We will introduce some of these projects and discuss why these do not meet the purpose of this thesis.

Schear el. al. developed a full featured trusted computing environment for cloud computing. They show in their paper how a TPM of a hypervisor can be virtualized and used by the guest operating system. This includes trusted bootstrapping, integrity monitoring, virtualization, compatibility with existing tools for fleet management and scalability[10]. The concept of a well known virtual environment does, however, not apply to our contribution. Furthermore, the the system should be self contained as good as possible and it should be possible to get information about the system via anonymous attestation.

- What exists in the field?
- Keylime - DONE
- Xaptum ECDAA
- FIDO 2 ECDAA
- Strongswan Attestation
- Linux IMA
- Secure Boot
- Intel TXT
- Trusted Execution Environment (TEE)
- nanovm (nanovms.com)

3 Concept

The theoretical tool that should be formed to one whole system implementation in this thesis.

3.1 Definition of the Biometric Sensor

What part fulfills the BS and what needs to be done. Record Sensor data, Network Discovery, send sensor data via trusted channel to PIA

3.1.1 What has the BS to do?

1. Listen for a Trigger to start the Authentication Process
2. Collect Sensor Data (Picture, Fingerprint) and calculate a biometric representation
3. Start Network Discovery and find the PIA of this person
4. Create a trusted and secure channel and transmit the attributes for verification
5. Restore the state of the system as it was before this transaction

3.2 Attack Vectors and Threat Model

3.2.1 The Threat Model

- Definition of sensitive data / privacy / metadata
- This version of BS is not owned by the user, there is no personal data in the System

- Rogue Personal Identity Agent (PIA)
- Metadata Extraction
- Attribute extraction
- Sensor Data Modification/manipulation
- Wiretap between Sensor and System (USB or network)
- Physical Manipulation of the BS-System
- Network - Retransmission of sensor data of a rogue BS
- Network - Blocking Data transmission of a rogue BS
- Rogue BS Sensor Data aggregation
- Rogue BS Sensor data modification before transmission

3.3 Trust and Security

Trust is an essential term in this thesis. In the world of IT security, the term *trusted computing* defines a secured environment where special or confidential computing jobs are dispatched. This environment or product usually meets the following requirements

- *Minimalization.* The number of features and hence the complexity must be as low as possible.
- *Sound definitions.* Every function should be well defined. There should be no margin for interpretation left. Security Engineers should be involved in the development.
- *Complete testing.* Testing for trusted computing includes a threat analysis and exhaustive testing if possible.

Since software and hardware testing is never complete, it is hard to find a good balance between feature set and testing completeness.

However trust in IT is not equal to security. It defines a subset of IT security where this small well defined environment is embedded in a larger system which is usually untrusted. Claiming a system *secure* spans the constraints of trust over the complete system, which is not affordable for commodity computers these days. However it is possible to use the trusted environment to get

some guarantees on the untrusted parts of a system as well In Chapter 3 we will show how trust will be extended in a commodity PC.

Differentiation between trust and security — and the problem that not everyone is using that right.

3.4 Systems of Trust

All trust systems are built on the standards of Trusted Computing Group.

3.4.1 Secure Boot, TXT, ...

Trusted Boot is not the same as Secure Boot. Explain the difference

3.4.2 TPM1.2

Initial Version of the crypto-coprocessor, successfully spread into many systems, but hardly any integration in Trust/security Software

3.4.3 TPM2.0

The *Trusted Platform Module* (TPM) is a small cryptocoprocessor that introduces a variety of new features to the platform. This module is part of a standard developed by the Trusted Computing Group (TCG), which current revision is 2.0[13].

The hardware itself is strongly defined by the standard and comes in the following flavors:

- *Dedicated device.* The TPM chip is mounted on a small board with a connector. The user can plug it into a compatible compute platform. This gives most control to the end user since it is easy to disable trusted computing or switch to another TPM.
- *Mounted device.* The dedicated chip is directly mounted on the target mainboard. Therefore any hardware modification is impossible. However most PC platforms provide BIOS features to control the TPM.

- *Firmware TPM (fTPM).* This variant was introduced with the TPM2.0 Revision. Firmware means in this context an extension of the CPU instruction set which provides the features of a TPM. Both Intel and AMD provide this extension for their platforms for several years now. When activating this feature on BIOS level, all features of Trusted Computing are available to the user.
- *TPM Simulator.* For testing reasons, it is possible to install a TPM simulator. It provides basically every feature of a TPM but cannot be used outside the operating system. Features like Trusted Boot or in hardware persisted keys are not available.

Even the dedicated devices are small microcontrollers that run the TPM features in software which gives the manufacturer the possibility to update their TPMs in the field. FTPMs will be updated with the Microcode updates of the CPU manufacturers.

The combination of well constrained hardware and features, an interface for updates and well defined software interfaces make TPMs trustworthy and reliable. Since TCG published the new standard in 2014 only six CVE-Entries handled vulnerabilities with TPMs[1]. Only two of them had impact on the implementation of a dedicated chip:

- *CVE-2017-15361*

Using the TPM

On top of the cryptographic hardware, the TCG provides several software interfaces for application developers:

- *System API (SAPI).* The SAPI is a basic API where the developer has to handle the resources within the application. However this API provides the full set of features.
- *Enhanced System API (ESAPI).* While still providing a complete feature set, the ESAPI makes some resources transparent to the application like session handling. Consequently, this API layer is built on top of the SAPI.
- *Feature API (FAPI).* This API layer is again built on top of the ESAPI. It provides a simple to use API but the feature set is also reduced to common use cases. Although the Interface was formally published from the beginning, an implementation is available since end of 2019.

[1] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=%22Trusted+Platform+Module%22

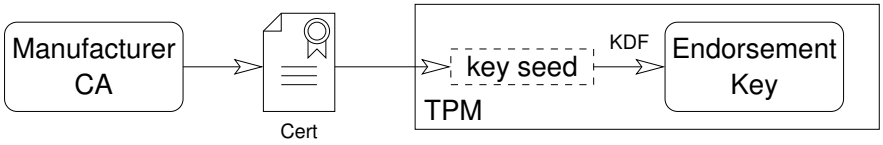


Figure 3.1: The manufacturer certifies every TPM it produces

The reference implementation of these APIs is published at Github[8] and is still under development. At the point of writing stable interfaces are available for C and C++, but other languages like Rust, Java, C# and others will be served in the future. The repository additionally provides the tpm2-tools toolset which provides the FAPI features to the command line. Unfortunately, the command line parameters changed several times during the major releases of tpm2-tools[9].

The Hardware

The TCG achieved with the previous mentioned software layers independence of the underlying hardware. Hence, TCG provided different flavors of of the TPM

TCG defined with the TPM2.0 standard a highly constrained hardware with a small feature set. It is a passive device with some volatile and non-volatile memory, which provides hardware acceleration for a small number of crypto algorithms. The standard allows to add some extra functionality to the device. However the TPMs used in this project provided just the minimal set of algorithms and also the minimal amount of memory.

Since TCG published its documents, several IT security teams investigated concept and implementations of TPMs.

- security problems with some implementations
- Hierarchies
- Endorsement Key
- Attestation Identity Key
- Key management

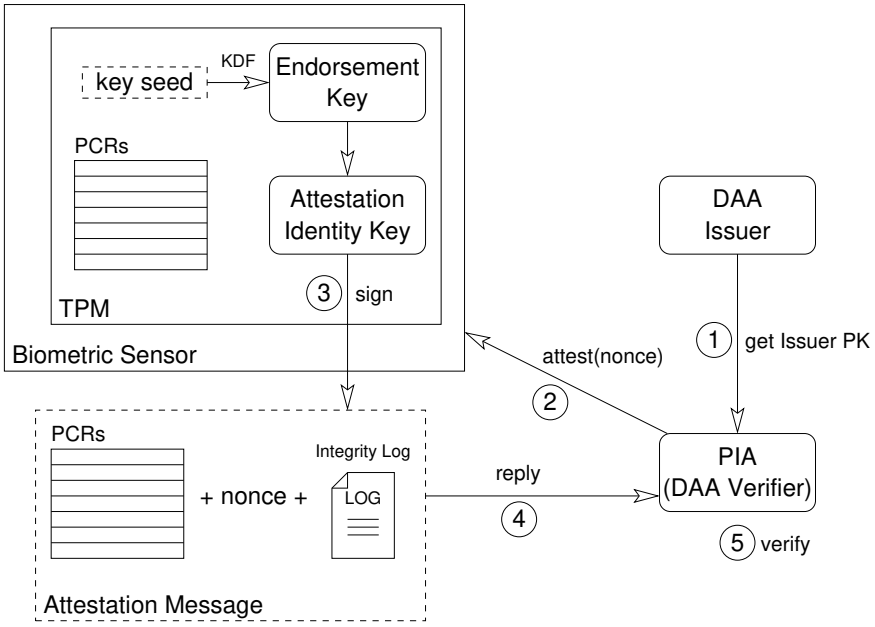


Figure 3.2: The DAA attestation process requires 5 steps. The PIA may trust the Biometric Sensor afterwards.

3.5 Trusted Boot

A boot process of modern platforms consists of several steps until the Operating System taking over the platform. During these early steps, the hardware components of the platform are initialized and some self tests are performed. This is controlled by either the BIOS (for legacy platforms) or the UEFI firmware.

TCG introduced in 2004 their first standard for trusted computing. As part in this standard, TCG defined a procedure, where every step in the early boot process is measured and saved in a *Platform Configuration Register* (PCR). The measuring part is a simple cryptographic extension function which works described in formula 3.1

$$\text{new\_PCR} = \mathit{hash}(\text{old\_PCR} \,||\, \text{data}) \tag{3.1}$$

The function of || represents a concatenation of two binary strings and the hash function is either SHA1 or SHA256. In recent TPM-platforms, both hashing algorithms are performed by default in each measurement. If there has to be measured more than one object in one PCR, the BIOS / UEFI has to perform the measurement in a deterministic way. The function allows this feature since the current value of the PCR is also part of the hash for the value. This feature is called *hash chaining* and ensures with a deterministic measurement procedure, that the resulting values are

always comparable as long as the measured components keep unchanged. The procedure of measuring the boot process did not change over the years and is still vaild for the most recent TPM2.0 standard.

A TPM has at least 24 PCR registers in the PC platform. Every PCR represents a different part of the platform. When TCG introduced Trusted Boot in 2004, UEFI was not yet available for the ordinary PC platform. Consequently, TCG standardized the roles of every PCR only for the BIOS platform. Later, when UEFI became popular, the PCR descriptions got adopted for the new platform. The most recent description of the registers, as defined in section 2.3.3 of the *TCG PC Client Platform Firmware Profile*[12], is shown in table 3.1.

Table 3.1: Usage of PCRs during an UEFI trusted boot process

PCR	*Explanation*
0	SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers
1	Host platform configuration
2	UEFI driver and application code
3	UEFI driver and application configuration and data
4	UEFI Boot Manager Code and Boot Attempts
5	Boot Manager Code Configuration and Data and GPT / Partition Table
6	Host Platform Manufacturer specific
7	Secure Boot Policy
8-15	Defined for use by the static OS
16	Debug
17-23	Application

The standard furthermore defines which part of the platform or firmware has to perform the measurement. Since the TPM itself is a purely passive element in the platform, the BIOS / UEFI firmware itself has to initiate the measurement beginning by the binary representation of the firmware itself. This procedure is well defined in the TCG standard and the platform user has to *trust* the manufacturer, that it is performed as expected. It is called the *Static Root of Trust for Measurement* (SRTM) and is described in section 2.2 of the TCG PC Client Platform Firmware Profile[12].

The SRTM is a small immutable piece of the firmware which is executed by default after the platform was reset. It is the first software that is executed on the platform and measures itself into PCR[0]. It furthermore must measure all platform initialization code like embedded drivers, host platform firmware, etc. as they are provided as part of the PC motherboard. If these measurements cannot be performed, the chain of trust is broken and consequently the platform cannot be trusted.

One may see a zeroed PCR[0] or a value representing a hashed string of zeros as a strong indicator of a broken chain of trust.

As the manufacturer of the motherboards do not publish their firmware code, one may have to reverse engineer the firmware to prove correct implementation of the SRTM. This is the point where the platform user has to trust the manufacturer as well as the manufacturer of the TPM. The PCR[1-7] are then written by the motherboard firmware itself. As last step, the bootloader is measured into PCR[4] and PCR[5] and then executed. Consequently, the bootloader and the OS are then responsible for continuing the chain of trust for this platform.

3.6 Integrity Measurements

As described in the previous section, when the boot process is eventually finished, the OS is then responsible for extending the chain of trust. Given a valid trusted boot procedure, the binary representation of the kernel is already measured. Therefore the Kernel itself has the responsibility to keep track of everything happening on the platform from the OS point of view.

Soon after the first TPM standard was published, the *Integrity Measurement Architecture* (IMA) for the Linux Kernel was introduced. Since Kernel 3.7 it is possible to use all IMA features, when the compiler options of the Kernel are set correspondingly.

IMA

Extend the Chain of Trust beyond the boot process. The Kernel can measure many different types of Resources. What is a useful set of measurements

3.7 Verify Trust with DAA

3.7.1 Definitions

For the definition of the algorithm, some notations and definitions are summarized in the following. Greek letters denote a secret that is not known to the verifier whereas all other variable can be used to verify the desired properties. The symbol || is a concatenation of binary strings or binary representations of integers.

Given an integer q, $\mathbb{Z}_q$ denotes the ring of integers modulo q and $\mathbb{Z}_q^*$ denotes the multiplicative group modulo q[5].

3.7.2 Discrete Logarithm Problem

Given a cyclic group $G = \langle g \rangle$ of order n, the discrete logarithm of $y \in G$ to the base g is the smallest positive integer x satisfying $g^\alpha = y$ if this x exists. For sufficiently large n and properly chosen G and g, it is infeasible to compute the reverse $\alpha = \log_g y$. This problem is known as *Discrete Logarithm Problem* and is the basis for the following cryptographic algorithms.

3.7.3 Signatures of Knowledge

Camenisch and Stadler[5] describe an efficient scheme for proving knowledge of a secret without providing it. They assume a collision resistant hash function $\mathcal{H} : \{0,1\}^* \longrightarrow \{0,1\}^k$ for signature creation. Furthermore, the algorithm is based on the Schnorr signature scheme[11]. For instance,

$$SPK\{(\alpha) : y = g^\alpha\}(m)$$

denotes a proof of knowledge of the secret α, which is embedded in the signature of message m. The one-way protocol consists of three procedures:

1. *Setup.* Let m be a message to be signed, α be a secret and $y := g^\alpha$ be the corresponding public representation.

2. *Sign.* Choose a random number r and create the signature tuple (c, s) as

$$c := \mathcal{H}(m \,||\, y \,||\, g \,||\, g^r) \quad \text{and} \quad s := r - c\alpha \pmod{\text{n}}\,.$$

3. *Verify.* The verifier knows the values of y and g, as they are usually public. The message m comes with the signature values c and s. She computes the value

$$c' := \mathcal{H}(m \,||\, y \,||\, g \,||\, g^s y^c) \quad \text{and verifies, that} \quad c' = c\,.$$

The verification holds since

$$g^s y^c = g^r g^{-c\alpha} g^{c\alpha} = g^r\,.$$

Camenisch and Stadler[5] state, that this scheme is extensible to proof knowledge of arbitrary number of secrets. Furthermore, complex relations between secret and public values can be represented with that scheme.

3.7.4 Bilinear Maps

The Camenisch-Lysyanskaya (CL) Signature Scheme[4] is used for the DAA-Protocol. Furthermore, the CL-Scheme itself is based on Bilinear Maps.

Consider three groups $\mathbb{G}_1, \mathbb{G}_2$, with their corresponding base points g_1, g_2, and $\mathbb{G}_T$. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ that satisfies three properties[3, 4]:

- *Bilinearity.* For all $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$, for all $a, b \in \mathbb{Z} : e(P^a, Q^b) = e(P, Q)^{ab}$.
- *Non-degeneracy.* For all generators $g1 \in \mathbb{G}_1, g2 \in \mathbb{G}_2 : e(g_1, g_2)$ generates $\mathbb{G}_T$.
- *Efficiency.* There exists an efficient algorithm that outputs the bilinear group $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ and an efficient algorithm for computing e.

3.7.5 Camenisch-Lysyanskaya Signature Scheme

The Camenisch-Lysyanskaya (CL) Signature Scheme...[4] allows efficient proofs for signature posession and is the basis for the DAA scheme discussed in section XY. It is based on a bilinear group $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ that is available to all steps in the protocol.

- *Setup.* Choose $x \leftarrow \mathbb{Z}_q$ and $y \leftarrow \mathbb{Z}_q$ at random. Set the secret key $sk \leftarrow (x, y)$ and the public key $pk \leftarrow (g_2^x, g_2^y) = (X, Y)$
- *Sign.* Given a message m, and the secret key sk, choose a at random and output the signature $\sigma \leftarrow (a, a^y, a^{x+xym}) = (a, b, c)$
- *Verify.* Given message m, signature σ and public key pk, verify, that $a \neq 1_{\mathbb{G}_1}$, $e(a, Y) = e(b, g_2)$ and $e(a, X) \cdot e(b, X)^m = e(c, g_2)$

Camenisch et al. stated in section 4.2 of their paper[3] that one has to verify the equation against $e(g_1, b)$ and $e(g_1, c)$ which is proven wrong here.

3.7.6 DAA History

Direct Anonymous Attestation (DAA) is a cryptographic protocol, which aims to provide evidence that a device is a honest member of a group without providing any identification information. Brickell, Camenisch and Chen[2] introduce DAA and implement the protocol for the TPM 1.2 standard. However it supports only RSA and has limitations in verifying attestation signatures. Hence, DAA is not used with the TPM 1.2 standard.

Since the DAA protocol is quite complex, it is difficult to provide a sound security model for DAA and formally proof the security properties of it. Chen, Morissey and Smart[7] add linkability to the protocol. Their approach for a formal proof is not correct, since a trivial key can be used for pass verification[3]

Camenisch, Drijvers and Lehmann[3] developed a DAA scheme for the new TPM 2.0 standard. It supports linkability and the proofs for security and correctness still hold. Furthermore, RSA and ECC cryptography is supported which makes it practicable for a wider variety of use cases. However, Camenisch et. al.[6] proposed a fix in the TPM 2.0 API to guarantee all requirements necessary for DAA. Xaptum implemented this DAA-variant including the fixes in the TPM API. The implementation will be discussed in Chapter 4.

Analyzing the security and integrity of this scheme would exceed the scope of this thesis. Hence this thesis describes the DAA protocol and assumes the correctness and integrity.

3.7.7 DAA Protocol

DAA is a group signature protocol, which aims to reveal no additional information about the signing host. According to Camenisch et al.[3] the DAA protocol consists of three parties.

- *Issuer* $\mathcal{I}$. The issuer maintains a group and has evidence of hosts that are members in this group.
- *Host* $\mathcal{H}$. The Host creates a platform with the corresponding TPM $\mathcal{M}\dot{\text{M}}$embership of groups are maintained by the TPM.
- *Verifier* $\mathcal{V}$. A verifier can check whether a Host with its TPM is in a group or not. Besides the group membership, no additional information is provided.

A certificate authority $\mathcal{F}_{ca}$ is providing a certificate for the issuer itself. `bsn` and `nym` Session ids sid is already available with communication session on the network or on the link between host and TPM. $\mathcal{L}$ is the list of registered group members which is maintained by $\mathcal{I}$.

- *Setup.* During Setup $\mathcal{I}$is generating the issuer secret key isk and the corresponding issuer public key ipk. The public key is published and assumed to be known to everyone.

  1. On input SETUP $\mathcal{I}$
     - generates $x, y \leftarrow \mathbb{Z}_q$ and sets $isk = (x.y)$ and $ipk \leftarrow (g_2^x, g_2^y) = (X, Y)$. Initialize $\mathcal{L} \leftarrow \emptyset$,
     - generates a prove $\pi \xleftarrow{\$} SPK\{(x, y) : X = g_2^x \wedge Y = g_2^y\}$ that the key pair is well formed,
     - registers the public key (X, y, π) at $\mathcal{F}_{ca}$ and stores the secret key,
     - outputs SETUPDONE

- *Join.* When a platform, consisting of host $\mathcal{H}_j$ and TPM $\mathcal{M}_i$, wants to become a member of the issuer's group, it joins the group by authenticating to the issuer $\mathcal{I}$.

  1. On input JOIN, host $\mathcal{H}_j$ sends the message JOIN to $\mathcal{I}$.
  2. $\mathcal{I}$ upon receiving JOIN from $\mathcal{H}_j$, chooses a fresh nonce $n \leftarrow \{0,1\}^\tau$ and sends it back to $\mathcal{H}_j$.
  3. $\mathcal{H}_j$ upon receiving n from $\mathcal{I}$, forwards n to $\mathcal{M}_i$
  4. $\mathcal{M}_i$ generates the secret key:
     - Check, that no completed key record exists. Otherwise, it is already a member of that group.
     - Choose $gsk \xleftarrow{\$} \mathbb{Z}_q$ and store the key as $(gsk, \perp)$.
     - Set $Q \leftarrow g_1^{gsk}$ and compute $\pi_1 \xleftarrow{\$} SPK\{(gsk) : Q = g_1^{gsk}\}(n)$.
     - Return (Q, π_1) to $\mathcal{H}_j$.
  5. $\mathcal{H}_j$ forwards JOINPROCEED(Q, π_1) to $\mathcal{I}$.
  6. $\mathcal{I}$ upon input JOINPROCEED(Q, π_1) creates the CL-credential:
     - Verify that π_1 is correct.

- Add $\mathcal{M}_i$ to $\mathcal{L}$.
- Choose $r \xleftarrow{\$} \mathbb{Z}_q$ and compute $a \leftarrow g_1^r$, $b \leftarrow a^y$, $c \leftarrow a^x \cdot Q^{rxy}$, $d \leftarrow Q^{ry}$.
- Create the prove $\pi_2 \xleftarrow{\$} SPK\{(t) : b = g_1^t \wedge d = Q^t\}$.
- Send APPEND(a, b, c, d, π_2) to $\mathcal{H}_j$

7. $\mathcal{H}_j$ upon receiving APPEND(a, b, c, d, π_2)
   - verifies, that $a \neq 1$, $e(a, Y) = e(b, g_2)$ and $e(c, g_2) = e(a \cdot d, X)$.
   - forwards (b, d, π_2) to $\mathcal{M}_i$.
8. $\mathcal{M}_i$ receives (b, d, π_2) and verifies π_2. The join is completed after the record is extended to $(gsk, (b, d))$. $\mathcal{M}_i$ returns JOINED to $\mathcal{H}_j$.
9. $\mathcal{H}_j$ stores (a, b, c, d) and outputs JOINED.

- *Sign.* After joining the group, a host $\mathcal{H}_j$ and TPM $\mathcal{M}_i$ can sign a message m with respect to basename bsn.

1. $\mathcal{H}_j$ upon input SIGN(m, bsn) re-randomizes the CL-credential:
   - Retrieve the join record (a, b, c, d) and choose $r \xleftarrow{\$} \mathbb{Z}_q$. Set $(a', b', c', d') \leftarrow (a^r, b^r, c^r, d^r)$.
   - Send (m, bsn, r) to $\mathcal{M}_i$ and store (a', b', c', d').
2. $\mathcal{M}_i$ upon receiving (m, bsn, r)
   - checks, that a complete join record $(gsk, (b, d))$ exists, and
   - stores (m, bsn, r).
3. $\mathcal{M}_i$ completes the signature after it gets permission to do so.
   - Retrieve group record $(gsk, (b, d))$ and message record (m, bsn, r).
   - Compute $b' \leftarrow b^r$, $d' \leftarrow d^r$.
   - If $\mathsf{bsn} = \perp$ set $\mathsf{nym} \leftarrow \perp$ and compute $\pi \xleftarrow{\$} SPK\{(gsk) : d' = b'^{gsk}\}(m, \mathsf{bsn})$
   - If $\mathsf{bsn} \neq \perp$ set $\mathsf{nym} \leftarrow H_1(\mathsf{bsn})^{gsk}$ and compute $\pi \xleftarrow{\$} SPK\{(gsk) : \mathsf{nym} = H_1(\mathsf{bsn})^{gsk} \wedge d' = b'^{gsk}\}(m, \mathsf{bsn})$.
   - Send (π, nym) to $\mathcal{H}_j$.

4. $\mathcal{H}_j$ assembles the signature $\sigma \leftarrow (a', b', c', d', \pi, \mathsf{nym})$ and outputs SIGNATURE(σ)

- *Verify.* Given a signed message, everyone can check whether the signature with respect to bsn is valid and the signer is member of this group. Furthermore a revocation list RL holds the private keys of corrupted TPMs, whose signatures are no longer accepted.

  1. $\mathcal{V}$ upon input VERIFY(m, bsn, σ)
     - parses $\sigma \leftarrow (a, b, c, d, \pi, \mathsf{nym})$,
     - verifies π with respect to (m, bsn) and nymif bsn $\neq \perp$.
     - checks, that $a \neq 1$, $b \neq 1$ $e(a, Y) = e(b, g_2)$ and $e(c, g_2) = e(a \cdot d, X)$,
     - checks, that for every $gsk_i \in \mathsf{RL} : b^{gsk_i} \neq d$,
     - sets $f \leftarrow 1$ if all test pass, otherwise $f \leftarrow 0$, and
     - outputs VERIFIED(f).

- *Link.* After proving validity of the signature, the verifier can test, whether two different messages with the same basename bsn $\neq \perp$ are generated from the same TPM.

  1. $\mathcal{V}$ on input LINK(σ, m, σ', m', bsn) verifies the signatures and compares the pseudonyms contained in σ, σ':
     - Check, that bsn $\neq \perp$ and that both signatures σ, σ' are valid.
     - Parse the signatures $\sigma \leftarrow (a, b, c, d, \pi, \mathsf{nym})$, $\sigma' \leftarrow (a', b', c', d', \pi', \mathsf{nym}')$
     - If nym = nym′, set $f \leftarrow 1$, otherwise $f \leftarrow 0$.
     - Output LINK(f)

4 Implementation

4.1 Trusted Boot

- Trusted Boot with GRUB 2.04: TPM support available; PCR mapping
- Secure Boot with Unified Kernel; another PCR mapping
- Benefits and Drawbacks of both variants

Limitations due to bad implementation on BIOS-Level, no Certificate Verification Infrastructure available for TPMs? Needs to be proven for correctness.

4.2 Integrity Measurement Architecture

Available on Ubuntu, RedHat and optionally Gentoo. The Kernel has the correct compile options set.

4.2.1 Handling external hardware

How can camera and fingerprint sensor be trusted? What is the limitation of this solution?

4.3 Interaction with TPM2

tpm2-tools 4.x are usable to interact with the TPM from the command line. Available on all major releases after summer 2019. Fallback is using the TPM2 ESAPI or SAPI, which is available on almost all Linux distributions.

4.4 Direct Anonymous Attestation

DAA Project from Xaptum: Working DAA handshake and possible TPM integration. Requires an Attestation Key which is secured with a password policy.

5 Conclusion and Outlook

5.1 Testing

These are the test results

5.2 Limitations

Still hard to set up a system like that. Documentation is available, but hardly any implementations for DAA and IMA.

5.3 Outlook

Hardening of the system beyond IMA useful. Minimization also useful, because the logging gets shorter.

Table 5.1 is an example of a table, in which the numbers are aligned at the comma, every second line is colored and the commands `\toprule`, `\midrule` and `\bottomrule` are used [1].

Table 5.1: Example

Länge l in m	Breite b in m	Höhe h in m
12.454	1.24	335.3
543.22	32.123	33.21
353.0	33.0	33.0
23.3	333.2	32.4

List of Figures

List of Tables

Bibliography

[1] Will Arthur, David Challener, and Kenneth Goldman. *A Practical Guide to TPM 2.0*. Jan. 2015. doi: 10.1007/978-1-4302-6584-9.

[2] Brickell, Camenisch, and Chen. "Direct Anonymous Attestation". In: *SIGSAC: 11th ACM Conference on Computer and Communications Security*. ACM SIGSAC, 2004.

[3] Jan Camenisch, Manu Drijvers, and Anja Lehmann. "Universally Composable Direct Anonymous Attestation". In: vol. 9615. Mar. 2016, pp. 234–264. isbn: 978-3-662-49386-1. doi: 10.1007/978-3-662-49387-8_10.

[4] Jan Camenisch and Anna Lysyanskaya. "Signature Schemes and Anonymous Credentials from Bilinear Maps". In: vol. 3152/2004. Aug. 2004, pp. 56–72. doi: 10.1007/978-3-540-28628-8_4.

[5] Jan Camenisch and Markus Stadler. "Efficient Group Signature Schemes for Large Groups". In: *CRYPTO '97* 1296 (Jan. 1997).

[6] Jan Camenisch et al. "One TPM to Bind Them All: Fixing TPM 2.0 for Provably Secure Anonymous Attestation". In: May 2017, pp. 901–920. doi: 10.1109/SP.2017.22.

[7] Liqun Chen, Dan Page, and Nigel Smart. "On the Design and Implementation of an Efficient DAA Scheme". In: Nov. 2010, pp. 223–237. doi: 10.1007/978-3-642-12510-2_16.

[8] TPM2 Software Community. *TPM2 Tools*. 2020. url: https://github.com/tpm2-software/tpm2-tools (visited on 05/15/2020).

[9] Pawit Pornkitprasan. *Its certainly annoying that TPM2-Tools like to change their command line parameters*. Oct. 2019. url: https://medium.com/@pawitp/its-certainly-annoying-that-tpm2-tools-like-to-change-their-command-line-parameters-d5d0f4351206 (visited on 02/27/2020).

[10] Nabil Schear et al. "Bootstrapping and Maintaining Trust in the Cloud". In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACSAC '16. Los Angeles, California, USA: Association for Computing Machinery, 2016, pp. 65–77. isbn: 9781450347716. doi: 10.1145/2991079.2991104. url: https://doi.org/10.1145/2991079.2991104.

[11] Claus Schnorr. "Efficient signature generation by smart cards". In: *Journal of Cryptology* 4 (Jan. 1991), pp. 161–174. DOI: 10.1007/BF00196725.

[12] *TCG PC Client Platform Firmware Profile Specification Revision 1.04*. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_PCClientSpecPlat_TPM_2p0_1p04_pub.pdf (visited on 08/01/2020).

[13] *The TPM Library Specification*. 2019. URL: https://trustedcomputinggroup.org/resource/tpm-library-specification/ (visited on 05/16/2020).

Installation instructions

1 Installing IMA on Arch

https://wiki.archlinux.org/index.php/Kernel/Arch_Build_System in combination with https://wiki.gentoo.org/wiki/Integrity_Measurement_Architecture:

```
sudo pacman -S asp base-devel
cd ~
mkdir build && cd build
asp update linux
asp export linux #Linux repo exported to this directory
```

Change *pkgbase* in `linux/PKGBUILD` to custom name, e.g. linux-ima. Check `linux/config` for the following settings:

```
CONFIG_INTEGRITY=y
CONFIG_IMA=y
CONFIG_IMA_MEASURE_PCR_IDX=10
CONFIG_IMA_LSM_RULES=y
CONFIG_INTEGRITY_SIGNATURE=y
CONFIG_IMA_APPRAISE=y
IMA_APPRAISE_BOOTPARAM=y
```

For optimizing file access, add to every fstab-entry *iversion*. It prevents creating a hash of the file at every access. Instead the hash will only be created when writing the file.

`updpkgsums` generates new checksums for the modified files.

`makepkg -s` then makes the new kernel

2 Installing Xaptum DAA

We use the Ubuntu 20.04 server edition for testing the environment. It supports Trusted Boot an IMA out of the box. Three systems need to be installed – the BS host, the issuer of the BS group and a verifier. Only the BS host needs to have a TPM in it, which requires a non-virtualized installation. The other hosts can easily be virtualized if needed.

Note: The DAA protocol can be tested without using the TPM.

2.1 Encrypted File System

Optional: It is usefult to enable disk encryption on the BS host. Therefore only the boot section remains unencrypted and the TPM is used to decrypt the disk.

2.2 Unified Boot Loader

2.3 TPM-tools

The TPM2-tools provide the features of the TPM to the shell and furthermore install the system API `apt install tpm2-tools`

2.4 Prerequisities for Xaptum ECDAA

Besides the building packages you should build two other projects from Xaptum. The first ist their variant of AMCL

```
1 sudo apt install cmake build-essential python3 python3-dev python3-pip gcc d
2 git clone https://github.com/xaptum/amcl.git
3 cd amcl
4 make
5 mkdir -p target/build
6 cd target/build
7 cmake -D CMAKE_INSTALL_PREFIX=/opt/amcl ../..
8 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
9 make
```

```
make test
make doc
sudo checkinstall
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./:/opt/amcl/lib
```

The Apache Milagro Crypto Library is now installed in `/opt`.

The next part is the `xaptum-tpm` project, which provides the interface between the ECDAA application and the TPM hardware.

```
git clone https://github.com/xaptum/xaptum-tpm.git
cd xaptum-tpm
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo -DCMAKE_INSTALL_PREFIX=/opt
cmake --build . --target install
```

2.5 Installing Xaptum ECDAA

Finally the main project can be installed:

```
git clone https://github.com/xaptum/ecdaa.git
cd ecdaa
mkdir build
cd build
cmake . -DECDAA_TPM_SUPPORT=ON -DCMAKE_INSTALL_PREFIX=/opt -DTEST_USE_TCP_TP
ctest -V
cmake --build . --target=install
```