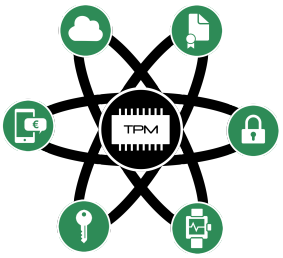

FutureTPM

D3.1

First Report on Security Models for the TPM

Project number:	779391
Project acronym:	**FutureTPM**
Project title:	Future Proofing the Connected World: A Quantum-Resistant Trusted Platform Module
Project Start Date:	1st January, 2018
Duration:	36 months
Programme:	H2020-DS-LEIT-2017
Deliverable Type:	Report
Reference Number:	DS-LEIT-779391 / D3.1 / v1.0
Workpackage:	WP 3
Due Date:	Sept. 2018 - M09
Actual Submission Date:	1st October, 2018
Responsible Organisation:	SURREY
Editor:	François Dupressoir
Dissemination Level:	PU
Revision:	v1.0
Abstract:	In this report, we review the FutureTPM requirements and identify effects on design and modelling targets and challenges. We then review the state of the art in threat and security modelling, in general and as applied to the TPM and other similar TEEs. We end the report by summarizing our findings, as well as planning and delimiting the research to be performed.
Keywords:	requirements, threat modelling, security modelling

The project FutureTPM has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779391.

Editor

François Dupressoir (SURREY)

Contributors (ordered according to beneficiary numbers)

SURREY, UBITECH, IBM, UB, UL, INESC-ID, UPRC, HWDU

Disclaimer

Executive Summary

In this report, we review the FutureTPM requirements and identify effects on design and modelling targets and challenges. We then review the state of the art in threat and security modelling, in general and as applied to the TPM and other similar TEEs. We end the report by summarizing our findings, as well as planning and delimiting the research to be performed.
We find that the FutureTPM requirements raise interesting design challenges such as the feasibility of a secure hybrid design that supports both classical- and quantum-secure cryptography from the same seeds. Further we identify that simulation-based and universally composable analysis methodologies are the most likely to usably scale to the whole TPM, but also identify research challenges to be overcome in order to precisely and securely capture security requirements for the whole of the TPM–in the presence of either classical adversaries or quantum-capable adversaries.
Finally, we identify some longer-term challenges that–although they do not directly contribute to the design and modelling of security for the FutureTPM–would serve as foundations in supporting future developments of a similar nature in a more principled way. Those include the development of robust foundations for quantum-UC, the consideration of low-level adversaries (for example, those with access to side-channels) in simulation-based and composable settings, and a principled formal treatment of remote attestation and trusted execution.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This report reviews the FutureTPM requirements identified in D1.1 FutureTPM Use Cases and System Requirements, known challenges in modelling security for the TPM, as well as the state of the art in threat and security modelling. This review aims at identifying engineering challenges and research gaps to be investigated during the course of the FutureTPM project, in order to ensure the successful delivery of the use case demonstrators. We also identify long-term research objectives of relevance to the project. In particular, we discuss challenges that may arise from the TPM's structure and specific deployment mode, and other challenges that may arise from the need to scale beyond even the largest published cryptographic security proofs.
The main outcome of this Deliverable is to delimit the scope of the security modelling and proof research to be performed during the course of the project. As such, we also discuss longer-term challenges in security modelling for hardware-assisted security and quantum-resistant cryptography, whose relevance to the use cases is less direct, but whose foundational value is clear.

1.1 Relation to other WPs and Deliverables

The review of existing literature presented in this report is guided by the requirements analysis performed in D1.1 FutureTPM Use Cases and System Requirements. This report complements D2.1 First Report on New QR Cryptographic Primitives, whose focus is on security modelling and concrete candidates for quantum-resistant cryptographic primitives, around which this WP will be designing a new Quantum-Resistant TPM. This report serves to guide research activities conducted as part of WP3, and will therefore feed directly into D3.2 Firt Report on the Security of the TPM and D3.3 Second Report on Security Models for the TPM. Finally, this report highlights interactions between threat modelling–which will be used in WP4–and security modelling, and performs a preliminary review of threat modelling techniques that could be used to identify reasonable trust and threat models for the FutureTPM in the use cases considered.

1.2 Deliverable Structure

We first analyse the functionalities of the TPM that will be exercised by our use cases, as well as their interdependencies (Chapter 2). We then identify the main modelling challenges that arise from these functionalities, and from the TPM as a whole (Chapter 3). We then review the state of the art in threat modelling and security modelling in general (Chapter 4), and as applied to the TPM and other hardware roots of trust (Chapter 5). This allows us to identify research gaps, that will need filled in order to meet the objective of defining security models for the TPM

functionalities seen as a whole, but also to make progress towards obtaining security proofs in such models (Chapter 6). Finally, we conclude by summarising the findings of our literature and technology review and planning activities for the next reporting period of WP3.

Chapter 2

Analysis of Requirements

In this Chapter, we recall the use case requirements identified in D1.1 FutureTPM Use Cases and System Requirements, and map them to TPM functionalities, as standardized in TPM 2.0, and summarized in the overview of TPM functionalities given in D1.1 FutureTPM Use Cases and System Requirements. Although not strictly related to security modelling, this requirements analysis is necessary to the design tasks of WP3, which will themselves heavily influence the requirements placed on the security model and analysis.

2.1 FutureTPM Requirements and TPM Functionalities

In D1.1 FutureTPM Use Cases and System Requirements, the Consortium devised a Minimum Viable Product (MVP) for the FutureTPM project. The MVP includes 37 technical requirements to be implemented by the FutureTPM. In this section, we recall these requirements (in the order shown in Table 11 (D1.1, Section 6.2)), and discuss the TPM functionalities that will be used to implement these requirements. We adhere to the Trusted Platform Module Library standard nomenclature, as defined by the TCG to refer to TPM functionalities.

Mandatory Requirements

1 **(TR.1.1.3) It should support protocol and algorithm agility.**
TCG maintains a registry for the protocols and algorithms supported under TPM 2.0. Another registry is maintained pertaining to the supported curves for Elliptic Curve Cryptography (ECC). Each entry has an associated value that can be used as a parameter to TPM commands, to control which cryptographic primitive is used to realise the command. Therefore, enlarging the set of supported algorithms may affect the implementation of: session commands; object commands; duplication commands; asymmetric and symmetric primitives; hash/HMAC/event sequences; attestation commands; ephemeral EC keys; signing and signature verification; command audit; integrity collection (PCR); enhanced authorization (EA) commands; hierarchy commands; miscellaneous management functions; and capability commands. The introduction of new cryptographic primitives may impose the need to maintain further registries. For instance, isogeny-based ECC is supported on finite fields with an order that is the square of a prime number. A registry might be required to keep record of the supported primes.

2 **(TR.1.3.6) Allow support for some legacy primitives/protocols.**
FutureTPM will strive to adhere to the TPM 2.0 standard as much as possible. Since this standard includes algorithm agility, support for legacy primitives and protocols can be maintained

by keeping their entries in the TCG registry.

3 **(SR.1.2.3) Reach QS-Level 1.**
With the exception of Random Number Generators (RNGs), Dictionary Attack Functions, and Clocks and Timers, the whole TPM architecture will have to be reviewed in order to achieve QS-Level 1.

4 **(SR.1.4.1) Allow the protection of sensitive information.**
The TPM may store sensitive user information persistently in its Non-Volatile (NV) memory. Access to this information might be limited to users who know a specific password (through password sessions), a specific key (through HMAC sessions) or through more complex policies (through enhanced authorization). Since NV storage is limited and might wear out quickly, keys might alternatively be stored in a TPM, with access restrictions, which encrypt data stored in the user's disk. One might enforce this key not to be duplicated, thus only allowing the data to be decrypted when the TPM is present. Nevertheless, duplication to another TPM might be useful to allow for backups; or on an enterprise scenario, where regulation might require access to the worker's data in certain situations.

5 **(SR.1.4.2) It should be hard for an adversary to learn the secret information required for any action.**
EA allows for the combination of several policies. Through the usage of EA, access to secret information may be subject to password controls, HMACs, smart cards providing digital signatures, the state of the PCRs, among others. Each one of these possibilities provides distinct levels of security; and the combination of several of them makes it harder for an adversary to learn the secret information required to access the data.

6 **(SR.1.4.3) Credentials should be stored on user device and must be protected from eavesdropping/leakage.**
Access to credentials might be bound to a trusted platform state, by sealing the corresponding encryption key to a specific set of PCR values. When this kind of access control is put in place, one is guaranteed to have the credentials only available on a system that is in a trusted state– indicating that no tampering has occurred, and preventing eavesdropping.

7 **(TR.1.1.4) It should support enhanced authorization (EA).**
EA is a TPM capability that allows specific actions or tests to be required to access a certain entity. Entities have an associated policy that defines the conditions for its use. For example:
- A policy may limit the use of a key unless selected PCRs have specific values;
- A policy may not allow use of a key after a specific time;
- A policy may require that authorization to change an NV index be provided by two different entities.

While a policy might be arbitrarily complex, it is expressed as a statistically unique digest. In order to use its associated entity, a user creates a policy session. The TPM is then given a sequence of commands that modify the digest in the policy session. After executing all the commands of the policy, the policy session might be used as an authorization session. When the accumulated digest matches the one associated with the entity, access is granted.

8 **(TR.1.4.1) Provide support for at least one major OS (e.g. Linux).**
The code specific to an OS in a TPM implementation should mostly relate to the TPM Com-

mand Transmission Interface (TCTI), the TPM Access Broker (TAB), the Resource Manager (RM) and the device driver.

9 (SR.1.1.1) Pseudorandom number generator.
TPM 2.0 specifies an RNG module that is used as a source of randomness in the TPM. It is used to generate nonces, in key generation, and for randomness in signatures. It typically contains an entropy source and collector, a state register and a mixing function. The entropy collector removes the bias from the entropy source. State registers are updated with the collected entropy, which are input into the mixing function to produce the random numbers.

10 (SR.1.1.2) Key generation and storage functionalities.
The TPM 2.0 specifies two manners through which a TPM might generate keys. In the first, an ordinary key is produced by using the RNG to seed the computation. In the second, which pertains to the computation of primary keys, keys are derived from a seed value using an approved key derivation function (KDF). The seed is permanently stored on the TPM. Moreover, in the case of a cryptographic primitive having weak keys, the TPM should detect the generation of those keys, discard them, and restart the key generation process.
TPM 2.0 provides an NV memory module, which might be used to store data persistently.

11 (SR.1.1.3) Hash functions.
Hash functions may be used directly by external software or as the side effect of many TPM operations (e.g. PCR extension, digest computation). A TPM should implement an approved hash algorithm that has approximately the same security strength as its strongest asymmetric cryptographic primitive.

12 (SR.1.1.4) MAC.
TPMs currently implement HMAC as described in ISO/IEC 9797-2. HMACs might be used for authorization, to protect the integrity of command parameters/responses, to create tickets, as a pseudo-random function, or used directly by external software. MACs other than HMACs might be considered in the context of the FutureTPM project.

13 (SR.1.1.5) Symmetric Encryption.
The TPM uses symmetric encryption to protect the confidentiality of some command parameters; and to encrypt protected objects stored outside of it.

14 (SR.1.1.6) Digital Signatures.
A TPM uses digital signature for attestation (e.g. as part of certification, quoting, building policies) and identification.

15 (SR.1.2.1) Support for possible QR crypto candidates for each category (symmetric, asymmetric and DAA).
The TPM 2.0 architecture is expandable and provides the mechanisms to extend the list of supported algorithms to include QR cryptographic primitives.

16 (SR.1.2.2) QR support for signing, key exchange, attestation.
Through algorithmic agility it should be possible to provide the QR primitives supporting signing, key exchange and attestation.

17 (TR.1.2.1) It should be feasible to implement the chosen post-quantum algorithms on platforms with restricted memory, while providing an acceptable performance.
The TPM 2.0 architecture provides several mechanisms to deal with its restricted memory while maintaining usefulness. For instance, cryptographic methods are used to increase the effective memory of the TPM by using cryptographic methods to store protected objects outside shielded locations in a secure manner. The TPM might generate cryptographic proofs to prove that it created or checked an externally provided value, instead of storing this information internally. Another way in which memory usage is reduced is through context management, wherein, in order to support sharing among many application, the objects, sequences, and sessions used by an application may be loaded into the TPM when needed and saved when a different application is using the TPM. When updating cryptographic primitives to a QR setting, similar challenges may be faced, and the consortium might draw from similar techniques to reduce memory usage.

18 (TR.1.1.2) It should provide a small set of platform configuration registers (PCRs).
PCRs are a fundamental part of the TPM. They allow to verify the integrity of a log wherein events that might affect the security state of a platform are recorded. When additions are made to the log, the TPM receives a copy of the log entry or a digest of the data related to the log. The TPM appends the received data to the contents of a PCR; and stores the hash of the result in the same PCR. The TPM might attest to the value in a PCR, which, in turn, provides proof of the integrity of the log.
A TPM might support multiple banks of PCRs, each associated with a different hash algorithm. Each PCR is typically associated with a different stage of the platform evolution. For instance, one PCR might be dedicated to recording measurements of the BIOS, a second to the boot ROM, etc. In addition, certain PCRs may be reset, mostly for development purposes.

19 (SR.1.3.1) Support software measurement (PCR extend) and measurement reporting (Quote), using QR algorithms.
Through algorithm agility, it should be possible to support the QR hashing and signing primitives necessary to achieve QR software measurement and measurement reporting.

20 (SR.1.3.2) Support remote attestation functionalities.
A TPM might produce a signature over software/firmware measurements in a PCR using an attestation-key protected by the TPM. A remote entity may verify the signature and inspect the measurement log to gain confidence about the state of the user's machine.

21 (SR.1.1.7) Public key encryption and key exchange.
Currently the TPM provides support for public-key cryptographic primitives based on the intractability of factoring large numbers and on solving the discrete logarithm problem over elliptic curves. The FutureTPM will offer alternate QR cryptographic primitives that implement these functionalities.

22 (TR.1.1.1) It should provide non-volatile random-access memory (NVRAM) storage.
An NVRAM may be used by the TPM to store persistent state. Part of the memory space should be available for allocation and use by the platform and entities authorised by the TPM owner.

23 (TR.1.3.3) Development and testing of a software FutureTPM, including adequate sup-

port for the Trusted Software Stack (TSS).
The TSS corresponds to the software residing on a platform that supports the TPM. A software FutureTPM might prove useful in two ways: it accelerates the development of a proof-of-concept TPM, enabling the introduction of experimental features; and when implemented on a secure environment, such as ARM's TrustZone, it might provide the same security guarantees as a hardware TPM.

24 (SR.1.3.3) Support sealing and binding operations.
When a data object is sealed to a trusted state, it can only be accessed when PCRs contain values attesting to that platform state. Policy sessions can be bound to an entity, making the session-key depend on the authorization value of that entity.

25 (TR.1.2.2) The selected crypto algorithms can be implemented securely on an identified platform.
The FutureTPM consortium will select QR cryptographic primitives based on a number of characteristics, including the robustness of their implementation against physical attacks, and on their suitability for the platforms identified in the use-case scenarios.

26 (TR.1.3.1) Selected algorithms should be chosen in such a way that it is possible to enhance the performance of the cryptographic calculations with a small hardware co-processor.
The selection of the QR cryptographic primitives should target a small resource consumption.

27 (TR.1.3.1) Selected post-quantum cryptographic primitives should be chosen to allow for a maximum reuse of hardware accelerator engines.
The sharing of hardware accelerator engines might help in reduce the overall circuit area of the FutureTPM.

28 (TR.1.3.5) Development and testing of a hardware FutureTPM (evaluation board), including adequate support for the Trusted Software Stack (TSS).
A proof-of-concept hardware FutureTPM will be developed on an evaluation board, along with the necessary software infrastructure to support the targeted use-case.

29 (TR.1.3.6) Allow support for some legacy primitives/protocols.
Addressed in requirement 2 (Repeated in Table 11 of D1.1)

30 (TR.1.3.4) Development and testing of a virtual FutureTPM, including adequate support for the Trusted Software Stack (TSS).
A virtual FutureTPM corresponds to TPM instances provided to virtual machines (VMs) by a hypervisor. Each of the secrets handled by a vTPM is sealed to a physical TPM. If the process for creating the guest VM, the vTPM and the vTPMs manager is trusted, a virtual TPM extends the chain of trust rooted in the hardware TPM to the virtual machines. Software support to the virtual FutureTPM will be provided to demonstrate the targeted use-case.

31 (SR.1.2.4) Provide a crypto library with TPM-backed keys implementing TLS with QR algorithms.
While TLS 1.3 has not yet standardised the usage of QR cryptographic primitives, there are drafts on Quantum-Safe Hybrid (QSH) Key Exchange for Transport Layer Security (TLS) ver-

sion 1.3, that might underpin the usage of QR capabilities provided by the TPM.

32 (SR.1.1.8) Direct Anonymous Attestation (DAA) [for SW TPM].
DAA is a signature scheme that provides anonymous signatures (no two signatures from the same signer can be correlated) or pseudonymous signatures (signatures from the same signer can be correlated but the signer's identity cannot be retrieved). The FutureTPM aims at providing a QR DAA protocol implementation in software.

Desirable Requirements

33 (TR.2.1.1) The efficiency of FutureTPM primitives and protocols should be similar or better than the ones currently provided by TPM 2.0.
The FutureTPM consortium will aim at maintaining current TPM performance.

34 (TR.2.2.4) Developing a Python based API or library that works with Django REST framework and integrates the developed TPM.
Django REST framework is a toolkit to develop web APIs. By integrating the TPM within this framework, one might, for instance, require platform identification or require a platform to be in a trusted state before servicing it.

35 (TR.2.3.3) Easy to port to existing architecture.
The FutureTPM consortium will target at making the developed code easily portable.

36 (TR.2.2.3) Trying to integrate existing APIs with any type of TPM (QR or otherwise).
The FutureTPM will try to adhere to the TPM 2.0 standard as closely as possible; introducing changes only when required to support post-quantum security.

37 (TR.2.3.2) Easy to support on mobile and IoT devices.
First, a large share of the mobile device market is based on the ARM architecture, which might provide a trusted execution environment (namely through TrustZone). A software TPM might be implemented therein, ensuring the portability of applications that make use of the TPM to those platforms. Second, the projected hardware TPM will target low resource consumption, making it more suitable for IoT devices.

2.2 Analysis of Dependencies between TPM Functionalities

There are 5 basic functionalities that the TPM supports as identified by D1.1 in chapter 2.1 pages 3-5: cryptography, storage, authorization, attestation and privacy. The cryptography functionality is stand-alone, as it provides basic primitives for other entities to use, there are some internal dependencies within the cryptography category though and only one external:

- Internal
  - Key generation and derivation rely upon random number generation and message authentication codes in order to create secure keys.
  - Signing schemes require asymmetric cryptography in order to function properly.
  - Message authentication codes, need hash functions in order to generate the hashes that are then encrypted.

- External
  - Random number generation, requires an entropy collector in order to be cryptographically secure.

We also note that FutureTPM requires the TPM functionality of cryptographic algorithm agility in order to enable us to install new algorithms. Algorithm agility is defined as the ability of algorithms being added or subtracted from the specification without requiring that the entire specification be rewritten.
In terms of the storage functionalities, TPMs provide three kinds of storage: secure storage, non-volatile storage and platform configuration registers. Secure storage is what TPM facilitates in order to extend the existing storage that the chip has. In order to achieve this, it encrypts the data and stores them externally, that is why secure storage depends on cryptography and more specifically, symmetric cryptography. Non-volatile storage, on the other hand, is a local and persistent memory that can store very sensitive data that are access protected, so there is a dependency on the TPM protections mechanism as defined in the TPM 2.0 specification in chapter 10 [82]. Finally, platform configuration registers, are protected memory registers for storing integrity measurements or platform software state. This is achieved through checking hashes and hash chains of the underlying software and data, it is obvious that the cryptographic primitive of hashing is required. Summing up, the functionality of secure storage depends on: symmetric cryptography, TPM protections and hash functions.
The authorization functionalities are concerned with how platform software can prove its authorization to call functions of the internal TPM. There are three levels of authorization: password, HMAC and Enhanced Authorization. In the case of password authorization, a password is sent in clear with every command and it does not depend on any functionalities. The HMAC solution, depends on message authentication codes in order to authorize each command separately by authenticating them using a password. Finally the enhanced authorization, is built on top of HMAC authorization sessions, and besides being based on a password, this kind of authorization also depends on TPM state including PCR values, external devices such as fingerprint readers or smart cards. So we identified the following dependencies for authorization: message authentication codes, PCRs (platform configuration registers) and external authentication devices.
Attestation is one of the crucial services of a TPM. It is the process by which a platform reports in a trusted way the current status of its configuration. The report can include as much information as required. The basis of the attestation are the measurements recorded in PCRs. The registers can then be read to know the current status of the platform and also be signed to provide a secure report. The sum of the attestation functionality dependencies are: asymmetric cryptography/signing schemes and PCRs.
Privacy functionalities are used to prove to third parties (verifiers) that they are communicating with a genuine TPM. This is done with the use of a unique public/private certified key-pair known as the Endorsement Key (EK) that every TPM has. However, if this key is used to sign objects, it will enable verifiers to uniquely identify this TPM and link all transactions it makes. The current specification solves this issue by providing the TPM with the ability to create as many Attestation Identity Keys (AIKs) as the user wishes in order to provide anonymity to the TPM. AIKs are not the only solution, as there is the option of Direct Anonymous Attestation (DAA) that is a group-oriented signature scheme with the use of RSA or elliptic curves. All in all, the privacy functionalities depend on: asymmetric cryptography, key derivation (in order to produce the AIKs) and a Direct Anonymous Attestation implementation. In Table 2.1 we summarize all the dependencies between the identified TPM functionalities. In the special case of cryptography we denote the internal and external dependencies as aforementioned in the relevant functionality category.

<table>
<tr><th>Functionalities</th><th colspan="2">Dependencies</th></tr>
<tr><td rowspan="2">Cryptography</td><td>Internal</td><td>External</td></tr>
<tr><td>• Random Number Generation for strong and secure key generation.
• Message Authentication Codes for key generation and key derivation.
• Asymmetric Cryptography for digital signatures.
• Hash Functions for the functionality of Message Authentication Codes.</td><td>• Entropy Collection from a reliable and high entropy source, in order for random number generation to be secure.</td></tr>
<tr><td>Storage</td><td colspan="2">• Symmetric Cryptography for secure storage to encrypt sensitive data.
• TPM protections that will enforce proper access control on non-volatile memory.
• Hash Functions which are used for the calculation of the values in platform configuration registers.</td></tr>
<tr><td>Authorization</td><td colspan="2">• Message Authentication Codes that are used for the authentication of each command.
• PCR Storage that is used to prove the state of the machine while issuing commands.
• External Authentication Devices which will prove the identity of the user.</td></tr>
<tr><td>Attestation</td><td colspan="2">• Asymmetric Cryptography and Signing Schemes are needed to sign the values of PCRs in order to prove the valid state of the device.
• PCR Storage that hold hash values calculated on the state of the device.</td></tr>
<tr><td>Privacy</td><td colspan="2">• Asymmetric Cryptography that is used to sign messages to prove the authenticity of the TPM.
• Key Derivation which will be used to generate Attestation Identity Keys in order to ensure the privacy of the device.
• Direct Anonymous Attestation that replaces Attestation Identity Keys and also ensures the privacy of the device.</td></tr>
</table>

Table 2.1: Summary of TPM Functionality Dependencies

Chapter 3

Challenges in TPM Modelling

As briefly illustrated in the previous chapter, the use cases cover all TPM functionalities, and most of their interdependencies. This allows us to consider and discuss the challenges we will face in modelling threats, security and functionality for the TPM as a whole, as well as those we will face in modelling the security of protocols that involve multiple TPMs.

3.1 Trust Assumptions

Generally speaking, the TCG works under the assumption that TPMs should be trusted to withstand any software attack. From a security modelling point of view, this means that no TPM could ever be compromised by a software-based adversary (for example, by revealing its primary seed). In practice, however, TPMs will be deployed in situations where an adversary can mount physical attacks–including side-channel and fault attacks–and locally compromise one or several TPMs. It is important to ensure that such an adversary is not able to break the security or privacy properties of uncompromised TPMs. It is therefore important for us to consider various models of trust, especially when analysing the security of functionalities that involve multiple TPMs. In particular, we will need to consider the following models:

- The TPM is completely trusted;
- The TPM is partially trusted;
- The TPM is subverted.

Such models have recently been considered in formal analyses of the ECC-DAA protocols standardized in TPM 2.0. (See Chapter 4.)

3.2 Split Operations and Untrusted Hosts

The TPM is a low-cost and relatively slow chip, and has limited resources. Therefore, some operations, including cryptographic operations, are split between the TPM and the software of its host platform (referred to as a Host). Further, the TPM relies entirely on the Host to communicate with remote partners (for example, the privacy-CA or a remote attestation partner). This has a further effect on security models, and requires careful consideration of the models to enable fine-grained modelling of trust both for the TPM (as justified above) and the Host (which may be a compromised platform). For example, the DAA signing operation is split between the TPM and

the Host, with the TPM being trusted for both security and privacy, but the host being trusted only to preserve its own privacy.
Defining security models for TPM functionalities will therefore require careful consideration of the roles played by different parts of the TPM (including its hardware component and various components of the TSS), and careful consideration of the appropriate trust assumptions on each of them. Investigations into refined trust models for split parties will further require careful interactions with the vulnerability analysis being carried out in WP4.

3.3 State, Command and Key Sharing

High-level TPM functionalities often require multiple calls to TPM commands. Due to its limited resources, the TPM itself rarely maintains state between calls to the commands. This has in the past been a source of vulnerabilities–that are made harder to find and analyze due to hidden invariants. Modelling such split functionalities may require a thorough understanding of these hidden invariants and of all expected uses of the TPM commands being captured by the model.
Since a single command could appear in multiple high-level functionalities–including sharing not only code, but also cryptographic material–analyzing the security of the TPM *as a whole* is critical. In particular, local models for specific functionalities do retain value but need to be carefully reworked to allow their use in more complex scenarios where part of the computation and the limited state may be shared with other functionalities that are being accessed concurrently.

3.4 Flexible and Secure Usage Policies

The TPM serves as hardware support for secure software systems, and is therefore meant to be used by external parties. If some of its security properties *must* hold independently of the usage that is made of it (in particular, properties of the Roots of Trust), properties of user objects stored on the TPM, or of interactions between users and the TPM, can only be guaranteed under certain conditions, including assumptions of trust in the user (for a particular scenario), and assumptions that a secure policy of interaction with the TPM was defined and followed. At both extremes, modelling security of the TPM is straightforward: a TPM that forbids any external interaction is easy to model and most certainly secure, but prevents any rigorous analysis of the security user applications; at the other end of the spectrum, a TPM that allows all interactions is also relatively easy to model, but provides only very limited security guarantees. Ensuring that our security models are both *realizable* in practice and *usable* to support security analysis for applications that use the TPM will require the careful definition of families of models parameterized by usage policies.

3.5 Multi-Tenant Security

In practice, TPMs–especially hardware TPMs used to back security in virtualized environments–will be multi-tenanted, and used by multiple users concurrently. Each of these users may have a different trust and authorization relationship with the TPM, including various authorization policies and external authentication factors. Capturing the security requirements of multi-tenanted scenarios into cryptographic security models is a complex challenge that currently remains open.

Chapter 4

Threat and Security Modelling

In this Chapter, we review and critically discuss threat and security modelling techniques in general.

4.1 Threat Modelling

This section provides an overview of the literature regarding the most known threat models and threat modelling approaches used and serves as a high level reference point. It is not specifically dedicated to threat modelling in TPM 2.0, this is the reason why it should be combined with other documents. Threat model is basically a high level description of how the attacker could exploit the possible vulnerabilities on the application environment. Threat modelling is designed as an aligned and systematic approach to detect all possible threats in the early phase of software systems and it is used to protect systems from vulnerabilities [63]. More specifically, it is a structured procedure for identifying and categorizing threats, and enumerating threat scenarios, which requires in-depth understanding of the architecture and underlying technology stack. One work that focuses on threat modelling and TPM 2.0 is [91]. This section presents a threat model for TPM 2.0 constructed using Microsoft's security development lifecycle threat modelling tool[1] and the STRIDE model. This preliminary task, albeit based on simple scenarios, highlights some potential pitfalls that should be considered when conducting further research into the applications of TPM. Based on different context, threat modelling approaches can be broadly classified into three main categories [60]: Asset-centric, Attacker-centric and Software-centric threat modelling approaches. In addition, various techniques have been published for performing threat modelling. Below, we describe the most popular threat modelling techniques: STRIDE, Attack Tree, and Attack Libraries.

4.1.1 STRIDE

The STRIDE approach to threat modelling was invented by Loren Kohnfelder and Praerit Garg in 1999 [58], and introduced by Microsoft as part of its Trustworthy Computing Security Development Lifecycle. The acronym STRIDE comes from the used threat classification scheme (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges). The STRIDE process consists of decomposing an application into constituent components on data flow diagrams and analyzing each component against possible threats from

[1] https://www.microsoft.com/en-us/sdl/adopt/threatmodelling.aspx

different STRIDE categories, and mitigating the identified risks. The application design is represented as a Data Flow Diagram (DFD) consisting of Data Flows, Data Stores, Processes, Interactors and Trust Boundaries. The primary purpose of the STRIDE is to recognize the possible threats and gather the possible attacks or threats. The categorization is secondary issue and often users have to select one of multiple choices and do not worry about the right category. This technique helps in the enumeration of threats based on attack properties. For each of these attack properties there is set of security themes violated as illustrated in the following table:

Attack Property	Security Theme
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial-of-Service	Availability
Elevation of Privilege	Authorization

Table 4.1: Attacks and the security themes they violate.

Threats are enumerated by considering each attack property and its corresponding impacted security theme. The STRIDE approach has two variants: STRIDE-per-Element and STRIDE-per-Interaction. The Microsoft SDL threat modelling tool applies STRIDE-per-Interaction.
STRIDE is designed to identify threats; however the vulnerabilities and the management of the vulnerabilities' coverage is left to others. In particular, to achieve completeness, threat modelling must identify and design strong countermeasures for the identified threats. Taking this a step further, and documenting the appropriate implementation countermeasures is not a feature of this technique.

4.1.2 Attack Tree

Attack tree is a conceptual representation of possible attacks against an application through which threats are ascertained. Basically, the attacks against a system are represented in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes [63]. Attack trees designed to represent in a structured way possible attacks on a system from the perspective of an attacker. The root of an attack tree is the ultimate goal of the attacker and is further refined through AND/OR decomposition into activities needed to achieve the goal. The leaf nodes usually denote system vulnerabilities, that an attacker could exploit to execute the intended attack scenario. Following are the 3 simple steps to construct an attack tree:

- Identify the goals – each goal can be a separate attack tree, in case of large attack vector even sub-goals can be represented in separate tree structures.
- Identify the various categories of attacks required to accomplish the goals.
- If a generic attack tree library exists, it can be plugged into the attack tree being constructed.

Upon identification of the attacks to accomplish the goal, attributes of the attacks have to be considered. Details such as probability of the attack, cost of the attack, and countermeasures have

to be documented to achieve completeness. However, it is impossible to capture all those details, and adding such information will complicate the tree structure at the expense of readability. Moreover, this model is suited to providing a high-level representation of the attacks, but does not suit modelling of threats at a more granular level. The attack tree when used independently as a threat modelling technique does not yield the best results.

4.1.3 Attack Library

Attack Library is a collection of attacks for finding threats against the application being developed [63]. The structure can be lightweight or highly organized. This is another type of threat modelling technique available to identify threats by looking from an attacker's perspective. The idea is to provide as much details as possible for an attack type to help threat modellers or the developer community to understand the landscape of threats. Any threat modelling technique adopting the attacker's perspective is more of a checklist model, i.e. traverse the library of attacks applicable in the context of the application, analyse whether the threats are handled, and identify countermeasures. Organizations can either develop their own library or leverage formal lists or dictionaries published by the security community or consortium such as Open Web Application Security Project (OWASP), Common Weakness and Enumeration (CWE), Common Attack Pattern Enumeration and Classification (CAPEC) and WASC Threat Classification.

4.1.4 OWASP

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted. The OWASP is a worldwide project that focuses on making security-based decision-making easier by providing different projects and data. These include the OWASP TOP-10 threats list, the cheat sheet series or the security testing guide [77, 67]. OWASP-proposed security practices and methodologies are widely used for web application projects, and it is therefore a suitable candidate for usage with threat modelling. However, we cannot rely on OWASP as our only source of security information and data, since its data oes not generally fulfil scientific requirements of rigour, provenance and transparency.

4.1.5 CWE

Common Weakness Enumeration is a formal list or dictionary of common software weaknesses that can occur in software architecture, design, code or implementation that may lead to exploitable security vulnerabilities. More specifically, CWE has identified critical programming errors that may lead to software vulnerabilities. CWE serves as a standard measuring parameter for software security tools targeting these weaknesses. The purpose is to provide a common baseline standard for weakness identification, mitigation, and prevention efforts. Details provided in each CWE include: description, applicable platform, common consequences, demonstrative example, observed examples, and related attack patterns.

4.1.6 CAPEC

CAPEC is an attack library and taxonomy of known attacks maintained by MITRE Corporation from USA. It is sponsored by the U.S. Department of Homeland Security and is under active maintenance [64]. It takes the form of a publicly available, community-developed list of common attack

patterns along with a comprehensive schema and classification taxonomy. The CAPEC can give more results regarding attacks than STRIDE's security properties but the CAPEC techniques are far more complex [77]. Velez and Morana [87] state that the library is one of the best and truly comprehensive attack library available. MITRE's records shows that the library currently contains over 517 attack patterns that are available through different interfaces such as Google-based search or a CAPEC ID search. CAPEC is the only big and clear attack pattern library. Moreover, CAPEC can also be obtained as an XML formatted file, which can be used locally to support more complex analyses. In particular, this offers the possibility of combining CAPEC data with domain and system-specific knowledge, as well as other threat modelling techniques, without having to interact with a remote database or server. However, if aggregation of the CAPEC XML data with other standard attack libraries is possible, its large size makes it a time-consuming process [87].

4.1.7 WASC Threat Classification

The WASC Threat Classification is an attack library by the Web Application Security Consortium. Its latest version, v2.0, was released in 2010, and presents threats in grid view or tree view. It categorizes vulnerabilities depending on whether they arise during the design, implementation or deployment of a system [43].

4.1.8 FutureTPM Threat Modelling

FutureTPM will build on-top of existing upper models and will propose a holistic approach for risk management in use cases of Trusted Computing. A more thorough description of the FutureTPM threat model will be provided in D4.1 Threat Modelling and Risk Assessment Methodology.

4.2 Security Modelling

The threat modelling techniques discussed in Section 4.1, although they capture threats to the overall architecture as well as implementation threats and vulnerabilities on software and hardware, cannot be used to analyze the security of the cryptographic primitives, schemes and protocols used to implement TPM functionalities. To define the expected security and privacy properties of TPM functionalities, we will rely on–and may need to extend–standard techniques for modelling cryptographic security. We now review these standard techniques, which cover both computational and symbolic notions of cryptography.

Symbolic cryptography (as initiated by Dolev and Yao [53]) models cryptographic operations as constructors in an algebra of terms (rather than functions on bitstrings), and restrict the adversary to only be able to act as specified by a particular set of functions and equational theories in order to gain knowledge or attempt to violate security properties. Symbolic models of security are often said to model cryptography as *perfect* and unbreakable, and are often well-suited to proving that a protocol does not misuse its cryptographic primitives.

Conversely, computational notions of cryptography capture cryptography–in a much finer-grained way–as probabilistic computations; defining the adversary only by its complexity and its access to the system–often based on oracles whose specification forms part of the security model. There are three main families of computational models of cryptographic security: game-based models (Section 4.2.1), simulation-based models (Section 4.2.2) and composable models (such as universal composability), discussed in Section 4.2.3. We explore them first, before discussing symbolic notions of security (Section 4.2.4).

4.2.1 Game-Based Models

Goldwasser and Micali [55] led the development of the modern way of thinking and reasoning about the security of cryptographic systems, defining a theoretical framework for this task, based on techniques from complexity theory. Their framework allows the cryptographer to precisely define the cryptographic problem being solved, identifying what the adversary can do (or how she can interact with the system under study) and when she is deemed to have successfully broken its security. A potential solution (scheme or protocol) is then proposed for the problem, as a set of algorithm, and evidence for the security of the high-level protocol can then be provided in the form of a reduction: any adversary that breaks the scheme or protocol can also solve another–less complex–problem which is widely believed to be hard.
Game-based models of security are well-understood, and have evolved since their inception to capture more practical aspects of cryptographic security. In particular, security claims have moved away from the asymptotic claims that were initially stated in these models, and towards *concrete* security claims that support precise parameter selection [21, 72]. In addition, some progress has been made by the *real-world cryptography* movement towards capturing implementation issues in proofs, instead of simply focusing on algorithms, leaving complex and often dangerous implementation details to non-implementers [69, 49, 27].
Despite their advantages, game-base models of security suffer from three major drawbacks, that are especially problematic when considering applications to cryptographic objects and constructions of the scale and complexity of the FutureTPM:

Inflexibility game-based security definitions often embed in the games themselves the condition the adversary has to meet in order to claim a cryptographic break; this makes it difficult, in cases where such conditions may depend on the application or on user-defined policies, to clearly list all conditions that may lead to a cryptographic break;

Lack of intuition game-based security definitions are often unintuitive an difficult to interpret. This makes it difficult to understand if a security property-as stated and proved–is indeed the property that is needed in order to secure a given application;

Non-composability game-based security definitions are highly non-composable: composing two game-based secure constructions to form a third is almost never guaranteed to yield a secure construction.

Mechanizing Game-Based Proofs As the best understood methodology for provable security, and thanks to its developments towards more practical security, game-based definitions and reasoning have enjoyed the most scrutiny from the formal methods community apart from the symbolic models we discuss in Section 4.2.4. First efforts in the direction of formal computational proofs in the computational model led to the development of CryptoVerif [26], a tool that automatically or semi-automatically searches for sequences of games and probability bounds that form a security reduction. Although CryptoVerif has enjoyed high profile successes–with applications to SSH [32] and, more recently, TLS1.3 [24]–its focus on providing a semi-automated verification solution makes it somewhat inflexible in modelling new protocols, or formalising new kinds of proofs.
CertiCrypt [14] is a library for the Coq proof assistant [2], whose goal is to support the manual formalisation–and automated machine-checking–of code-based game-playing proofs. Unlike previous efforts in using Coq to formalize computational security proofs, its goal is to design

[2] https://coq.inria.fr

and develop a framework that supports the systematic construction of machine-checked security proofs. Being a Coq library means it enjoys the same strong soundness guarantees that Coq offers, but makes it difficult to develop, refine and use without deep expertise in interactive theorem proving and type theory. This leads to a lack of modularity in proofs that makes CertiCrypt proofs even less composable than pen-and-paper game-based proofs, and severely limits its applicability to large proofs, although it was used to verify security proofs for simple primitives [93, 15].
EasyCrypt [12, 11] is a fresh implementation of the programme logics implemented in CertiCrypt that is not encumbered by Coq's limitations, and also makes use of SMT solvers to discharge simple goals resulting from proof verification. The additional flexibility offered by EasyCrypt allows the tool to better support abstract and modular reasoning, and the development of formal proofs for more complex objects, with formal game-based security proofs for complex asymmetric schemes such as OAEP [13] and PSS [10], and larger protocols such as authenticated key exchange [9] and electronic voting protocols [45, 46].
FCF [70] reimplements CertiCrypt's programme logics for reasoning about game-based proofs in Coq, leaving some of its more complex aspects (for example, the obligation to formally prove complexity bounds on constructed adversaries) informal. CryptHOL [19] is a framework for formalizing game-based cryptographic proofs in the Isabelle-HOL proof assistant. Neither tool has been used extensively in practice.

4.2.2 Simulation-Based Models

Following from the inflexibility and lack of intuition of game-based definitions, Goldreich, Micali and Wigderson [54] propose a new paradigm for specifying cryptographic security. Rather than specifying a precise condition the adversary has to break as part of the game which also bounds her interactions with the system, simulation-based definitions use a game to represent the allowed interactions, and an *ideal functionality*–often expressed as a simple protocol relying on a trusted third-party–to represent the adversary's goal: distinguishing a concrete construction from the ideal functionality. This technique has several advantages over game-based definitions of security.
First, the real world-ideal world paradigm allows for very flexible definitions of cryptographic security: it is not necessary (although it is useful) for the ideal functionality to be "secure" (for some separately-defined notion of security). In particular, simulation-based claims simply state that the real construction is "equivalent from the point of view of the adversary" to some idealized construction. In general, the security of such ideal functionalities is clear from construction. (For example, in the case of Multi-Party Computation [54], it is simply a trusted third-party that communicates with all other parties via private channels; in the case of pseudo-randomness, the ideal functionality will often be a truly random equivalent.) In some cases, the ideal functionality could have insecure modes of operation, allowing for some insecurity, but also supporting a more detailed analysis of these sources of insecurity on a simpler object that does not involve cryptography.
Second, in most cases, the ideal functionality by itself gives a reasonably clear intuition of what it means for the construction to be secure–not only as a proof goal, but also as a usable definition of security that can be use to analyze applications that make use of the construction.
This improvement to the intuition of security definition also takes some steps towards compositional reasoning about security–and particularly supports the modular construction of provably-secure schemes in layers.
These advantages come at a cost, however: proofs in the simulation-based setting are often much more involved than in the game-based setting. This also makes such proofs more difficult

to trust.

Mechanizing Simulation-Based Proofs Although efforts in mechanizing game-based security proofs have yielded significant results, the same issues that arise from pen-and-paper proofs still apply, and prevent precise reasoning about more complex cryptographic functionalities, or more complex constructions.
As discussed, some of the tools developed for the mechanical verification of game-based proofs (for example, EasyCrypt [11], but also FCF [70]) afford some modularity in the construction of security proofs–even those that are game-based. This is due to the fact that the programming language and logic techniques used in those tools can in fact be used almost in the same way to formalize simulation-based cryptographic notions and proofs. For example, EasyCrypt was used without modification to produce simulation-based proofs for 2-Party and Multi-Party Computation [3, 57]–quintessentially simulation-based cryptographic functionalities–and for some secret-sharing-based side-channel countermeasures whose security is defined in a simulation-based setting [8].
However, if those tools can indeed capture simulation-based notions and do support the formal verification of proofs for realizations of those security functionalities, the proof techniques they use are not always adapted to ease the development of those formal proofs. The F* programming language [81] is a dependently-typed functional programming language that has been used extensively to prove simulation-based properties of cryptographic protocols–including very high-profile formal proofs (and attacks) on TLS1.2 and TLS1.3 [52]. Their security modelling approach, based on ideal functionalities and simulations, is very close to the basic principles of simulation-based security (and have inspired new perspectives on simulation-based cryptography, such as state-separating proofs [31]) and afford the same advantages (modular construction of proofs) while also supporting the automatic verification of side-conditions to composition theorems. However, the approach suffers from the same drawbacks as traditional simulation-base cryptographic techniques and cannot easily support parallel compositions with shared state–requiring the use of hand-proved meta-theorems. In addition, some of the standard simulation-based abstractions for basic cryptographic primitives cannot be expressed readily in their language.

4.2.3 Composable Models

If simulation-based notions of security take some steps towards compositional approaches to cryptographic security, they are often focused on modularity rather than general composition. In particular, simulation-based definitions rarely capture concurrent compositions of protocols and constructions that may share some state or underlying oracle.
Universal Composability [38] (UC) is a framework that supports the specification of almost any notion of cryptographic security in a unified an systematic way, and in which security is maintained under a general ”protocol composition” operation, called universal composition. In this model, in addition to the construction, adversary, ideal functionality and simulator, an environment is used as part of the specification of security, and a construction is said to be UC-secure whenever, for any computationally-bounded environment and any computationally bounded adversary, the environment cannot distinguish between the “real” world, where the adversary interacts with the construction, and an “ideal” world, where the adversary interacts with a simulator–which itself interacts with the ideal functionality. The simple addition of the environment allows one to reason about the security of the protocol in any context, and enables support for more general composition–including arbitrary concurrent instances of a protocol.

Still, if this framework provides very strong security guarantees, obtaining trustworthy proofs of UC security is quite complex, and even modelling–regardless of proofs–complex cryptographic objects such as the FutureTPM would be a significant undertaking, as it would require modelling–to the finest detail–all possible interactions between an idealized TPM and its environment.
The original paper [38] proposing UC as a framework is regularly updated to reflect new flavours of the framework, and it was still updated as recently as 2017. We now discuss some of the extensions to UC that are of most potential value to the FutureTPM project.

Responsive Environments

The original UC framework [38] does not distinguish between interactions that are part of the protocol, and interactions that are part of the problem specification and are meant to provide meta-information to protocol parties (for example, the algorithm to be executed by a party, or messages meant to signify dynamic corruption). For the latter kind of messages, it would often be reasonable to assume that the adversary (or environment) responds immediately to such meta-messages and releases control back to the communicating party. However, the original UC framework is highly *non-responsive*. Although it is possible to artificially enforce responsiveness, this still requires the protocol designer–as a security modeller–to understand the framework and why attacks that leverage these meta-messages are not admissible in practice.
Camenisch et al. [37] propose a variant of the UC framework that includes new concepts of *responsive environments adversaries*, which must provide valid responses to modelling-related queries before activating any new protocol or functionality. The concept of responsivity proposed by Camenisch et al. [37] is general, and applies to many prior variants of UC.

Global Setup

Cryptographic protocols are often designed and analyzed under some assumption that all participants have access to some global state that is trusted to have some basic security properties. For example, the common reference string (CRS) and random oracle (ROM) [20] models are such *global trusted setup* assumptions. The UC framework, however, fails to provide expected security guarantees in the present of such trusted setup, with some natural protocols that meet the strongest known composable security notions, and are still vulnerable to bad interactions with rogue protocol that use the same setup.
Canetti et al. [39] extend the notion of UC security to recover its original intuitive guarantees even in the presence of globally available setup. The new notion prevent bad interactions even in the presence of adaptively chosen protocols that share the global setup. However, this extension proves too strong, and realizing Zero Knowledge or commitment functionalities becomes provably impossible even in the CRS model.
In an attempt to circumvent these impossibility results, Canetti, Jain and Scafuro [40] propose a global–but non-programmable–random oracle in the Generalized UC framework, and show that some basic cryptographic primitives with composable security can be efficiently realized in their model.
Although this is a step towards recovering composability with trusted setup, the non-programmable nature of their random oracle prevents many practical protocols from being proven secure in their framework–for example, the canonical randomize-and-hash commitment scheme. Camenisch et al. [34] study alternative definitions of a global random oracle, and show that these allow the provably GUC-secure realization of a number of essential cryptographic primitives, including public-key encryption, non-committing encryption, commitments, Schnorr signatures, and hash-

and-invert signatures. Some of these results hold generically for any suitable scheme proven secure in the traditional ROM, whereas others hold for specific constructions only. The results include many highly practical protocols, for example, the folklore commitment scheme $H(m|r)$ (where m is a message, and r is the random opening information), which is far more efficient than the construction Canetti et al. [40] could prove secure.

Quantum UC

The question of whether Universal Composability-style results hold in the presence of a quantum adversary remains open. Unruh [86] first investigated it and proposes a quantum version of the UC model which enjoys the same compositionality guarantees. The framework is given first in terms of computational quantum UC security (restricted to BQP adversaries, environment, and simulator), and then lifted to the statistical quantum UC case. Unruh further shows that composition theorems analogous to the classical ones still work in this setting. Furthermore, it is shown that every statistically classically UC secure protocol is also statistically quantum UC secure. Such implications are not known for other quantum security definitions. As an application, the authors prove that in this model statistically quantum UC secure oblivious transfer protocols can be constructed from commitments and, as a corollary, we get that computationally quantum UC secure protocols for general multi-party computation can be constructed from commitments.

4.2.4 Symbolic Models

Symbolic models, also known as Dolev-Yao models from their initial formulation by Dolev an Yao [53] following Needham and Schroeder's seminal work [65], have been an object of study for more than 30 years. This family of models have demonstrated their usefulness in supporting the design of protocols, by developing confidence and trust in their security, and enabling the discovery of several attacks in widely-deployed protocol. The contents of this section are based on [1, 44, 47, 23], and we refer the reader to these works for a further discussion.

Symbolic Models: A Primer

Unlike the computational model of cryptography, in which messages are broken down into their constituent bits, and adversaries are probabilistic Turing machines (or similar low-level models of computation, in the case of composable models), symbolic models instead consider messages as elements in an algebra of terms, possibly refined by an equational theory. This essentially has the effect of specifying the adversary not by *how* it can compute, but by *what* it can compute.
For a set of function symbols $\mathcal{F}$, a set of names $\mathcal{N}$ and a set of variables $\mathcal{X}$, the set of valid terms $T(\mathcal{N}, \mathcal{X}, \mathcal{F})$ in that algebra is inductively defined as the set of names, variables, and function symbols applied to other terms. Cryptographic primitives are regarded as perfect black boxes modelled as function symbols in that algebra. Messages exchanged by the different parties in a protocol are terms on these primitives (instead of bit strings), and the inference rules define which messages can be computed (or derived) from an a priori given set of messages. Security properties are also modelled formally.
As a simple example, consider the inference system shown in Figure 4.1, that contains 5 derivability rules corresponding to a symmetric encryption scheme, where x and y are terms.
The function symbols senc and conc denote symmetric encryption and (injective) concatenation, respectively. This simple set of derivability rules encode that, for example, given two terms x and y, the adversary can derive their concatenation $\mathsf{conc}(x, y)$ (rule CONCI). Conversely, derivation

$$\frac{x \quad y}{\mathsf{conc}(x,y)}\ (\textsc{ConcI}) \qquad \frac{\mathsf{conc}(x,y)}{x}\ (\textsc{ConcE-l}) \qquad \frac{\mathsf{conc}(x,y)}{y}\ (\textsc{ConcE-r})$$

$$\frac{x \quad y}{\mathsf{senc}(x,y)}\ (\textsc{Senc}) \qquad \frac{\mathsf{senc}(x,y) \quad y}{x}\ (\textsc{Sdec})$$

Figure 4.1: Derivability rules encoding adversary knowledge in a symmetric encryption scheme.

rules ConcE-l and ConcE-r let the adversary derive each of the two subterms of a concatenation from the concatenation itself (i.e. concatenation must be reversible). Rule Senc states that an adversary that has knowledge of two terms x and y can derive the encryption of x under y. Importantly, this model states (through the absence of a rule allowing the adversary to perform the operation) that there is no way to recover neither x nor y from $\mathsf{senc}(x,y)$ alone. However, the last derivation rule, Sdec, models the fundamental requirement that the message x can be recovered, given its encryption $\mathsf{senc}(x,y)$ and the key y.
Inference systems such as the above model what an attacker can compute. However, the set of inference rules does not always suffice to accurately model cryptographic primitives (and what the adversary can learn from them). For example, in cases where the adversary gains information not by learning new values but by observing the difference between two behaviours. For example, the bitwise XOR operator or modular exponentiation are standard cryptographic primitives that cannot be properly modelled by an inference system alone. Modern symbolic models therefore extend the term algebra with a set of equations. For example, in the inference system above, symmetric decryption can be modelled by introducing an explicit destructor sdec through the following straightforward equation.

$$\mathsf{sdec}(\mathsf{senc}(x,y),y) = x.$$

In symbolic models, it is assumed that the adversary has access to all public information, and can actively interfere with the communication between honest parties by blocking, intercepting, modifying, delaying, or injecting messages . This is informally stated as “the adversary is the network.” Any message sent by an honest party is assumed to have been received by the adversary, and then dropped or forwarded to another party, possibly after modification. Also, any message received is assumed to have come from the adversary. Furthermore, the adversary can impersonate any party, as identities are assumed to be public. Of course, the adversary is restricted to only compute terms that can be derived from the set of terms that are publicly known using the rules of the specified term algebra and equational theory. The adversary can neither manipulate the encryption’s bit representations nor guess ground terms.
Though less realistic than the computational model, the symbolic model is more manageable, and well suited to show that a protocol is broken under the given assumptions about the attackers capabilities. Most importantly, it makes it easier to build automatic verification tools.

Comparison with Computational Models

By treating cryptography and network messages as abstractly as they do, symbolic models introduce an additional gap between model and reality. As such, security proofs in symbolic models provide weaker real-world guarantees. If symbolic attacks often translate into practical real-world attacks on protocols, symbolic proofs of security may very well miss practical attacks due to the additional abstraction.

In some cases, however, it can be shown, via *computational soundness* results for particular symbolic models (for example, starting with Abadi and Rogaway's seminal work [1]), that symbolic guarantees imply computational guarantees under somewhat reasonable assumptions on the cryptography. In most cases, however, fulfilling these additional assumptions (for example, that encryption hides plaintext length) entails a practical cost that protocol designers are not willing to take on.

Symbolic Models for Mechanized Security Reasoning

A paramount question in the field of symbolic modelling consists in determining if an adversary can derive a certain term t, given an equational theory E and the set of terms S that she observes. This *derivability* property is often denoted as $S \vdash_E t$. The derivability problem is undecidable in the general case, as the adversary can build an infinite set of terms: he can build terms of unbounded size, and the protocol can be executed any number of times. However, the derivability problem is easily decidable (in PTIME) for a very particular class of inference systems called local inference systems. These systems satisfy that whenever $S \vdash_E t$, then there exists a derivation of t that only uses subterms in $S \cup \{t\}$.
There exist many automatic verification tools that can be used to evaluate the security of protocols under the symbolic model, and they take different approaches to solve the derivability problem.

Bounded Model-Checking Tools. By bounding both the size of messages that the adversary can generate and the number of executions of the protocol, it is possible to make the search space finite in order to apply standard model-checking techniques. This allows the technique to benefit from the fact that model-checking is fully automated an produces counterexamples if the system fails to satisfy the specified property–illustrating the failure with an attack. The following tools fall within this category.

FDR [66] This tool is a refinement checker for the communicating sequential processes (CSP) algebra. It allows processes to be defined in a machine-readable version of CSP, and is then able to check various assertions about these processes.

SATMC [84, 6] The SAT-based Model-Checker for Security Protocols and Security-sensitive Applications (SATMC) stems from a successful combination of encoding techniques originally developed for planning with techniques developed for the analysis of reactive systems. It has been successfully applied in a variety of application domains (security protocols, security-sensitive business processes, and cryptographic APIs) and for different purposes (design-time security analysis and security testing). SATMC strikes a balance between general purpose model checkers and security protocol analyzers. It provides a number of distinguishing features, including the ability to check the protocol against complex temporal properties (e.g. fair exchange) or analyze protocols (e.g. browser-based protocols) that assume messages are carried over secure channels (e.g. SSL/TLS channels).

General Model-Checking Tools. These tools bound only the number of executions of the protocol, and allow terms of arbitrary size to be manipulated by the adversary. Derivability in this case is generally decidable–albeit NP-complete. The following tools fall within this category.

CL-AtSE [83] The main idea in the Constraint Logic-based ATtack SEarcher (CL-AtSE) consists in running the protocol or set of services in all possible ways by representing families of

traces with positive or negative constraints on the adversary knowledge, on variable values, on sets, etc. Thus, each run of a service step consists in adding new constraints on the current intruder and environment state, reducing these constraints down to a normalized form for which satisfiability is easily decidable, and decide whether some security property has been violated up to this point. CL-AtSE does not limit the service in any way except for bounding the maximal number of times a protocol can be iterated, in the case such an iteration is allowed in the specification. Otherwise, the analysis might be non-terminating on secure services and only heuristics, approximations, or restrictions on the input language could lift this limitation.

If a security property of the input specification is violated then CL-AtSE outputs a warning (UNSAFE), some details about the analysis (e.g. whether the considered model is a typed or an untyped one), the property that was violated (secrecy, for instance), statistics on the number of explored states, and, finally, an ATTACK TRACE that gives a detailed account of the attack scenario. If no attack was found then similar information is provided (but the ATTACK TRACE).

OFMC [18] The On-the-fly model checker (OFMC) combines two ideas for analyzing security protocols based on lazy, demand-driven search. The first is the use of lazy data types as a simple way of building efficient on-the-fly model checkers for protocols with very large, or even infinite, state spaces. The second is the integration of symbolic techniques and optimizations for modelling a lazy Dolev–Yao adversary whose actions are generated in a demand-driven way. The tool has proven useful to find all known attacks and discovered a new one in a test suite of 38 protocols from the Clark/Jacob library in a few seconds of CPU time for the entire suite.

Semi-Automated Verification. As discussed, the derivability problem becomes undecidable when both the message size and number of sessions is unbounded, so no fully-automated tool can perform symbolic analysis in the general case. However, other approaches–interactive or semi-automated–can be used to provide partial support–either through user hints (or full interaction), or through the use of a ternary Secure/Don't Know/Insecure output. The following tools have been used to perform symbolic analysis. Some of these tools are general-purpose tools.

Isabelle [85] Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. The main application is the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols.

Tamarin prover [16] The Tamarin prover is a security protocol verification tool that supports both falsification and unbounded verification in the symbolic model. Security protocols are specified as multiset rewriting systems and analysed with respect to (temporal) first-order properties and a message theory that models Diffie-Hellman exponentiation combined with a user-defined subterm-convergent rewriting theory.

TA4SP [17] The Tree automata based verification of security protocols (TA4SP) tool approximates the adversary knowledge by using regular tree languages and rewriting. For secrecy properties, TA4SP can show whether a protocol is flawed (by under-approximation) or whether it is safe for any number of sessions (by over-approximation).

Maude-NRL Protocol Analyzer (Maude-NPA) [71] Maude-NPA is an analysis tool for cryptographic protocols that takes into account many of the algebraic properties of crypto-systems that are not included in other tools. These include cancellation of encryption and decryption, Abelian groups (including exclusive-or), exponentiation, and homomorphic encryption. Maude-NPA uses an approach similar to the original NRL Protocol Analyzer; it is based on unification, and performs backwards search from a final state to determine whether or not it is reachable. Unlike the original NPA, it has a theoretical basis in rewriting logic and narrowing, and offers support for a wider basis of equational theories that includes commutative (C), associative-commutative (AC), and associative-commutative-identity (ACU) theories.

ProVerif [25] This protocol verifier is based on a representation of the protocol by Horn clauses. It can handle many different cryptographic primitives, including shared- and public-key cryptography (encryption and signatures), hash functions, and Diffie-Hellman key agreements, specified both as rewrite rules or as equations. It can handle an unbounded number of sessions of the protocol (even in parallel) and an unbounded message space. This result has been obtained thanks to some well-chosen approximations. This means that the verifier can give false attacks, but if it claims that the protocol satisfies some property, then the property is actually satisfied. The considered resolution algorithm terminates on a large class of protocols (the so-called "tagged" protocols). When the tool cannot prove a property, it tries to reconstruct an attack, that is, an execution trace of the protocol that falsifies the desired property.

StatVerif [73, 5] This automated verifier for security protocols extends ProVerif with constructions which may used to model security protocols with global mutable state. In particular, the kind of security properties which may be verified is the same as ProVerif, and the syntax of StatVerif processes is an extension of the syntax for ProVerif processes with constructs for modelling states.

Scyther [48] This tool is fully automatic, always terminates, and can provide three different results: verification for an unbounded number of sessions, attack, or verification for a bounded number of sessions. It supports only a fixed set of cryptographic primitives (symmetric and asymmetric encryption and signatures). It proves secrecy and authentication properties. A version named scyther-proof generates Isabelle proofs of security of the verified protocols.

We note that SATMC, CL-AtSE, OFMC and TA4SP are back ends of the AVISPA framework (AVISPA stands for Automated Validation of Internet Security Protocols). The AVISPA project aims at developing a push-button, industrial-strength technology for the analysis of large-scale Internet security-sensitive protocols and applications. It is a platform that offers different protocol verification back-ends. In addition to the four tools mentioned above, the AVISPA framework also comprises a translator tool, from a high-level specification language into a common intermediate format to all the verification tools.

4.2.5 Beyond Black-Box Security

So far, we have discussed techniques and tools for modelling and security proofs in *black-box* models, where the adversary has only limited access to a system–usually delimited by a set of oracles (in the computational setting) or a set of derivability rules (in the symbolic setting). Practical adversaries, however, may have access to much more information about a system than their input-output behaviour, for example through side-channels [61, 62] or fault attacks [28]. Models in which the adversary has such powers are often called *grey-box* models.

Although such attacks have been known to work in practice for a long time, formal models that support reasoning about security in their presence are relatively recent. The development of automated, semi-automated, and interactive tools to reason about cryptographic security has been partially motivated by–and is increasingly successful in–obtaining concrete security guarantees on *implementations* and in models that take side-channels and perhaps other low-level attacks into account. This aligns somewhat with the recent developments in real world cryptography [49] mentioned above.

Almeida et al. [2] present a definitional framework for extending game-based security definitions to account for potential leakage in any given model, and demonstrate how existing proofs for the black-box game-based security notion can be extended into the corresponding grey-box model by discharging only additional proof obligations about the leakage itself–rather than having to consider a brand new proof. As part of this methodology, which has been applied to various constructions in various grey-box models (focusing mainly on timing and power side-channels), the identification of a precise but reasonable model of what leaks during a cryptographic computation is crucial.

In the context of FutureTPM, this leakage model would depend on the kind of TPM being considered, with a software TPM possibly offering the contents of its whole memory to a compromised OS, and a hardware TPM being only vulnerable to the most advanced (and costly) physical attacks.

Chapter 5

Security modelling for TPM (and TEE)

We finally review existing (partial) security models for the TPM and other TEEs. We organise this review by the functional components they cover, then by the threat and security model they consider. We focus our review on literature that presents precise *security models* for the functionalities they analyze. In most cases, unfortunately, the TPM and the functionalities it is meant to provide have been modelled only informally, with little mathematical rigour. Many of the models that we did find are designed to support machine-checking, rather than pen-and-paper proofs. This gives our literature review an apparent bias towards formal methods, and reveals a significant gap in the research literature. This gap also explains the apparent dearth of deep analyses of TPM functionalities (other than those that require the development of novel cryptography, such as DAA): it is difficult to analyse the security of the TPM if its security objectives have never been clearly defined. We discuss this gap further in our discussion of the work to be performed as part of Work Package 3. Ryan [74] presents a short survey of automated analyses of select TPM1.2 functionalities, many of which were carried out in the symbolic model.

5.1 Cryptography, Storage and Key Management

At the core of the TPM lies its collection of cryptographic primitives, which both rely on and support its key hierarchies–including its storage hierarchy. This section reviews models and analyses of these two components.

Shao, Feng and Qin [75] develop a type-based framework to analyse the security of the TPM's Protected storage API in the same scenario where authorization is not used. They use it to show that (part of) the TPM API cannot be misused to extract from the TPM the value of keys whose FixedTPM attribute is set. The framework is defined to provide symbolic security guarantees, and is limited to a very small subset of the TPM's commands.

Zhang et al. [94] use Tamarin to model a large subset of the TPM's key management API, including key duplication and import, and search for proofs of symbolic secrecy and integrity for duplicated and imported blobs. They find simple attacks. In particular, they note that the raw TPM mechanisms cannot provide identification of the destination on duplication, or of the origin of a blob import, which allows for breaks in authentication and secrecy. These attacks, rather than identifying a break in the TPM specification, illustrate the need to consider fine-grained security policies in the security models of TPM components.

Wang et al. [89] describe a formal game-based model of the TPM's cryptographic support commands in CryptoVerif, and a CryptoVerif-based proof of the fact that honestly-generated keys can be used to securely encrypt or authenticate messages, even if the adversary can otherwise interact with the TPM, including to create his own keys and to request encryptions under user keys

(thereby modelling a worst-case scenario). Their model does not allow duplication, and considers a single TPM (and therefore does not consider problems related to the migration of keys).

5.2 Sessions and (Enhanced) Authorization

If the articles discussed above focus on modelling the core properties of the TPM's hierarchies, they abstract away the TPM's authorization mechanisms, which enforce usage control on TPM-protected objects. Instead, the articles above place themselves in a pessimistic scenario where all keys can be used by the adversary. If this is a good abstraction to analyse the security of the protected storage mechanism in isolation, it does very little to support the verification of *applications* that make use of that protected storage. In particular, such an abstraction lacks the flexibility needed to allow controlled usage of cryptographic functionalities in larger protocols. We now review existing models of the TPM's authorization mechanisms, for TPM1.2 and 2.0.

Chen and Ryan [42], in the first attempt at defining security for (part of) the TPM, model TPM1.2's authorization mechanisms and find that a naive treatment of the authentication data (authdata) used to access objects can lead to complete usurpation of the TPM's secure storage in multi-tenant scenarios, even when encrypted sessions are used. They also prove using ProVerif symbolic security properties (authentication and secrecy) of a modified version of the protocol, which is used as part of TPM2.0's Enhanced Authorization mechanism. Delaune et al. [51] propose more general notions of authentication for TPM1.2's session mechanism as properties of its API, and show that they are sufficient to capture known attacks and verify fixed APIs.

Wang, Qin and Feng [88] formalize TPM2.0's HMAC-based authorization sessions in CryptoVerif and prove that they provide the expected authenticity properties: that the TPM only executes protected commands when called by a user who possesses the appropriate secret, and that callers engaged in a protected sessions with a TPM can trust that execution of the commands indeed occurred on the TPM upon receiving results. The contributions are limited to only HMAC-based authorization sessions, and abstract away much of the complexity due to the TPM's session mechanisms being used for various purposes.

Shao et al. [76] produce a SAPIC model for a larger subset of TPM2.0's Enhanced Authorization mechanism, which includes a large subset of the policies (including PCR-based authorization, policies based on counters stored in NVRAM, signature-based authorization), but abstracts some of the object management away. They use Tamarin to obtain proofs of symbolic authentication for most modes of authorization, and identify some cases where misuse is possible and needs to be managed by careful usage of the TPM's API.

5.3 Direct Anonymous Attestation

Direct Anonymous Attestation is the only cryptographic functionality that was developed specifically for the TPM, which otherwise reuses standard cryptography wherever possible–although sometimes in novel and somewhat untested ways. As such, DAA–taken in isolation–has been the focus of much more attention than the rest of the TPM, leading to the development and refinement of complex models for its security. We describe this evolution below.

Brickell, Camenish and Chen [29] propose a security model of DAA, as well as its RSA-based realization for TPM1.2. The proposed model is game-based, and is therefore difficult to use in security proofs for larger systems that may involve parallel DAA instances, or compositions. A ProVerif-based analysis by Backes, Maffei and Unruh [7], which considers the protocol's security

properties (and in particular the authentication guarantees it is meant to provide) finds possible attacks on the join protocol, whereby anonymous credentials can be delivered to the wrong TPM. A proposal for ECC-based DAA [30]–a version of which was to be included in the TPM2.0 specification–was formalized in ProVerif and its privacy properties analyzed in the symbolic model under an additional symbolic assumption by Smyth, Ryan and Chen [78, 79]. This symbolic analysis was among the main drivers for the development of equivalence-based notions of symbolic security–needed to express strong secrecy properties (equivalent to IND-CPA) and privacy properties such as anonymity and unlinkability. This analysis, which focuses on privacy properties for which the host is trusted, is performed on a model of the cryptography that does not consider the fact that DAA operations in the real TPM are split between the Host and the TPM itself. Xi and Feng [90] formalize the TPM2.0 DAA-related APIs in ProVerif and verify that all expected properties of the API are indeed met. They also propose a new notion of privacy–*forward anonymity*–and show that, although it is not met by the API as specified–a small modification to the API enhances the protocol to meet it. In particular, this failure of forward anonymity is in fact related to a weakness that prevents reductions to the more standard assumptions [33].
However, these positive symbolic security results–by the very nature of the models they are obtained in–fail to capture some important realistic attacks. Indeed, Bernard et al. [22] show that existing models of security for DAA–including new simulation-based models introduced to analyze the ECC-based version of Brickell, Chen and Li [30]–can be used to deem secure protocols that are fully insecure. Bernard et al. [22] propose game-based security models for pre-DAA–usable in a setting where the Host is trusted. A more flexible and realistic trust model is recovered by Camenisch, Drijvers and Lehmann [35], who propose a UC-based security model for Direct Anonymous Attestation. This shift to UC-based security models further enabled refinements of the trust model, such as the consideration of the effect of compromised TPMs on the security of uncompromised TPMs [36]. Camenisch et al. [33] propose modifications to the specification of DAA in TPM2.0 that improves the overall security of TPM2.0 DAA (by weakening the assumption on which the proof relies), and prove their security–including forward anonymity–in a UC-based setting.

5.4 Applications of Trusted Computing

The main focus of the contributions discussed above was to develop security models for specific TPM functionalities, and provide evidence that the TPM as specified meets those definitions of security (or identify weaknesses). Although care is usually taken to ensure that the security models and specified functionalities can be used to construct secure applications that rely on TPM, those works discussed above do not discuss the suitability of the defined models to perform such analyses. In this Section, we provide an overview of published works on models developed to support this kind of analysis, both for general applications of trusted computing, and for specific high-profile applications–such as BitLocker.

5.4.1 Applications of TCG TPM

Gürgens et al. [56] develop an automata-based model of TPM1.2's functionalities–abstracting away its cryptography–and use simulation-based techniques to explore the state space to identify security and practicability issues in four TPM-based scenarios: secure boot, secure storage, remote attestation and data migration. The work focuses on defining and analysing the security of applications that use the TPM, but the authors' findings led to the identification of some

usability issues and the addition of guidance notes and recommendations to the TPM specification. The issues identified also guided design decisions during the development of the TPM2.0 specification.
Delaune et al. [50] propose a framework for the analysis of protocols that make use of the TPM's PCRs, and apply it to two PCR-based case studies: Microsoft's BitLocker use of sealing to a PCR value and a digital envelope protocol that allows a user to choose whether to perform a decryption, or to verifiably renounce the ability to perform said decryption. The models developed are application-specific, but do illustrate the use of cryptography-aware verification tools in applications where multiple TPM functionalities interact (in the case of BitLocker: PCRs, the storage hierarchy, and Enhanced Authorizations). Yu et al. [92] develop a formal model of TPM2.0's HMAC-based authorization, and formally analyze its application to a DRM scenario. They find that, although TPM2.0's mechanism resists some of the attacks identified in this DRM context on TPM1.2, other attacks are still present and require care on the application side.

5.4.2 Remote Attestation and Secure Execution

Fine-grained modelling of trust is needed to provide meaningful security guarantees in scenarios that involve multiple TPMs–for example when considering key migration. Establishing such fine-grained trust requires a remote attestation mechanism. Making use of it in security proofs and arguments requires models that capture attestation mechanisms as part of the functionality. As far as we are aware, no such models exist for the TPM itself–other than those developed by Gürgens et al. [56] for the verification of remote attestation applications (rather than for the verification of realisations of the underlying functionalities). We therefore review related literature for other Trusted Execution Environments (TEEs), such as Intel SGX. This also includes models for Trusted Execution.
Pass, Shi and Tramer [68] develop a formal model for a generic "attested execution" functionality, and explore the expressive power of such functionalities. Based on this formalisation, they obtain strong possibility and impossibility results on the kind of secure functionalities that can be realized based on such attested execution processors. Although their definitions of attested execution is in general not supported by the TPM, it may be possible to restrict the set of supported attested functionalities to capture the TPM and reuse some of their formalism and results. Similarly, Subramanyan et al. [80] present a Coq-based formalization of 'trusted abstract platforms' (TAPs) and formally show that TAPs implement secure remote execution, and that both Intel SGX and Sanctum–two processors that support enclaves and secure execution–are indeed TAPs. Again, the target of the formalization effort is not the TPM, but rather a TEE that does provide a secure execution mechanism similar to enclaves.

5.5 Towards Quantum-Resistant TPMs

Ando et al. [4] take steps towards making the TPM quantum-resistant by integrating a QR hash primitive into its design. In particular, they show how a stateful hash-based signature scheme could be integrated into existing TPM designs as a drop-in replacement for currently supported asymmetric signature algorithms using a reasonable amount of secure storage on the TPM to prevent rollback attacks, where the adversary could roll the state back to force the reuse of a one-time signing key. The paper presents security models for TPM cryptographic functionalities that offload some of the storage to main memory. These models may be interesting as a basis to further extend the security analysis in a quantum-resistant setting.

Chapter 6

Conclusions and Planned Research

In Chapter 5, we have reviewed existing security models for the TPM and its functionalities and applications. We have shown that a holistic security definition for the whole of the TPM functionalities has not been provided so far. In other words, although formal security definitions for some concrete TPM functionalities do exist, as well as for direct anonymous attestation protocols, currently there is no security definition for the TPM as a whole, i.e., there is no formal security definition for TPMs that describes all the security properties that a TPM must provide for each of its functionalities.
Therefore, the provision of the first holistic security definition for the TPM is one the major goals of the FutureTPM project. We note that such a security definition will be useful not only for the analysis of quantum-resistant TPMs, but also for the security analysis of the current TPM standard. Indeed, the security properties provided by the current TPM standard have mainly been analyzed by considering each of its cryptographic functionalities in isolation. However, as described in Chapter 2, interactions and interdependencies between the different functionalities provided by a TPM exist, and therefore the security objectives of a TPM must be defined by considering the TPM as a whole–and even considering multiple TPMs in some cases. Having this holistic TPM security definition will subsequently allow us to reason about the security of concrete TPM instantiations and to work towards providing a proof of their combined security.
In Section 6.1, we discuss the modelling techniques that can be used to create our TPM security definition. As pointed out already in Chapter 3, providing such a security definition, as well as analyzing and proving the security of a concrete TPM implementation, is not trivial. We discuss the main difficulties in Section 6.2. Finally, in Section 6.3, we discuss longer-term research challenges related to TPM security modelling.

6.1 Modelling techniques to be used for FutureTPM

In Chapter 4, we have reviewed threat and security modelling techniques. Although threat modelling (as reviewed in Section 4.1) is not directly relevant to the modelling challenges we face in developing a holistic security model for the TPM, it is important to precisely understand the threats each of the TPM instantiations will face in order to ensure that the developed security model precisely captures those properties applications may rely upon. In particular, the threat models identified in WP4 (using the techniques reviewed in Section 4.1 *should* inform the kind of trust models (for example, whether some hosts can be trusted, or if some TPMs must be considered insecure in specific scenarios) and leakage models (for example, in some settings, the adversary may have access to precise power consumption measurements that would be inaccessible in others) to be considered when defining what it means for a TPM to be secure.

In order to create our TPM security definition, the FutureTPM project could use a symbolic or a computational security model. As pointed out in Section 4.2, symbolic models of security are often said to model cryptography as *perfect* and unbreakable, and are often well-suited to proving that a protocol does not misuse its cryptographic primitives. A symbolic model does not imply a guarantee of security, due to the gap between the formal model and the concrete implementation of a protocol. Although an attack shown on the symbolic model directly translates to an attack in the computational model, the converse is not generally true. Computational models capture cryptography in a more fine-grained way as probabilistic computations and define the adversary only by its complexity and its access to the system, but proofs in those models are much more complicated. Our review of existing models for parts of the TPM shows that, where symbolic models are used, combinations of TPM functionalities can often be considered (although none of the models capture all functionalities at once). On the other hand, all studies carried out in a computational model of cryptography consider only a single functionality, without considering its interactions with any of the TPM's other parts–even when interactions are significant, such as between the Storage and Enhanced Authorization functionalities.

As described in Section 4.2, computational models can be game-based, simulation-based or composable. The main advantage of composable models is precisely that they guarantee security under composition, i.e., if the conditions specified in the model are fulfilled, once a protocol is proven to realize an ideal functionality for a certain task, the protocol will still provide the security properties defined by that ideal functionality when executed in arbitrary contexts. The composition property is particularly relevant to define security for TPMs. We expect TPMs to be used in different contexts and as building blocks of many different cryptographic protocols. Consequently, it is vital that the security analysis and proof of a TPM remains valid when used in all those contexts.

On the other hand, it must be remarked that the choice of one model to provide a TPM security definition does not exclude the use of other models to define the security of TPM building blocks. For example, consider that the security of a cryptographic task is defined in a composable model by an ideal functionality, and consider a protocol that uses several cryptographic primitives as building blocks. When analyzing the security of this cryptographic protocol and proving that it realizes that functionality, typically one assumes that those building blocks fulfil their respective security definitions, and these security definitions do not need to use a composable model like the TPM security definition, i.e., they can use the game-based or the simulation-based model. Therefore, it is likely that, when proving that a TPM fulfils our TPM security definition, we will use security definitions for its building blocks in any of the computational models.

In short, we will likely develop both symbolic and computational models, the former providing a quick way of testing theories an validating design choices on abstractions of the TPM, and the latter allowing for a more precise modelling of cryptographic functionalities. Our holistic security model for the TPM will likely be expressed in a composable framework (either UC or one of its variants), in order to more easily support reasoning about parallel compositions of TPMs and about the security of arbitrary applications. However, we will likely also develop simulation-based notions of security for parts of the TPM that can be considered independently. Indeed, simulation-based notions provide a way of reasoning about security in a modular way, are very lightweight compared to fully composable technique. In addition, there is significant tool support for the development of semi-automated and interactive proofs in simulation-based settings, which is currently lacking for fully composable models.

6.2 Main Research Challenges

With this in mind, there are significant research challenges to be faced before a full security model can be produced for the TPM, but also when designing the FutureTPM.

6.2.1 Design Challenges

First, we examine the Mandatory Requirements (as listed in Section 2.1 to identify research challenges that we will face in designing the FutureTPM to integrate quantum-resistant cryptography in its operations.

Secure Hybrid Hierarchies

Considering Mandatory Requirement 2, the FutureTPM should allow support for some classically-secure primitives and protocols. This poses an interesting design challenge if we are to keep the FutureTPM as close as possible to current designs while supporting both classically-secure and quantum-resistant cryptography: keys in the hierarchies are generated from seeds installed on the TPM during fabrication, that cannot–except for the Storage seed–be replaced. However, using the same seed as the root of trust for both classical and quantum-resistant hierarchies may become problematic if not carefully managed. Indeed, a quantum-capable adversary with platform access may be able to perform classical cryptographic operations based on the seed and extract its value from observed results.
Designing and analyzing–in isolation–the security of a hybrid hierarchy mechanism will be necessary in order to meet Mandatory Requirement 2 fully.

FutureTPM-backed Quantum-Resistant TLS

Mandatory Requirement 31 requires that the FutureTPM project ”provide a cryptographic library with TPM-backed keys implementing TLS with QR algorithms.” Although a draft exists for a quantum-safe version of TLS 1.3 exists, it may be necessary for members of the project to get involved in designing or refining proposals to ensure they are compatible with the choices of primitives made for FutureTPM, or at least to stay aware of the directions taken by the wider community of interest surrounding cryptographic standards.

6.2.2 Modelling Challenges

The task of defining the security of a TPM will face challenges of scale. However, it will also face concrete challenges related to the specific nature of the TPM.

Trust levels. As explained in Chapter 3, our TPM security definition will need to give the adversary the ability to corrupt a TPM. There can be different levels or states of corruption, from a fully corrupt TPM to a fully trusted TPM. The security definition must state the security properties that the TPM must provide under each of the states of corruption. Because a TPM provides a large number of cryptographic functionalities, we foresee that our security definition will need to consider the case in which the adversary corrupts one or more of the functionalities while the remaining ones remain uncorrupted. We note that some composable frameworks have specific provisions to model different levels of trust. For example, GNUC [59] defines trust hierarchies and hierarchical corruptions.

Split parties. As explained in Chapter 3, in some cryptographic protocols that use a TPM, the TPM is embedded into a host party that performs some operations on its behalf. In the security definition for that protocol, it is necessary to give the adversary the ability to (fully or partially) corrupt the host, while the TPM remains trusted. On the other hand, the host and the TPM should not be considered fully independent parties. The main reason is that typically, when two parties are independent, both of them have access to the network independently of each other, while a TPM relies on the host in order to be able to access the network and communicate with other parties. Consequently, the host and the TPM constitute a "split party". In order to model security for split parties, the FutureTPM project could follow some of the security definitions for DAA protocols, which are reviewed in Section 5.3.

Shared State. Some cryptographic functionalities provided by the TPM may share some information internally, such as secret keys or state information. Some of them may also use a common global setup. If two cryptographic protocols share state, and their security has been proven considering each of them in isolation, their security properties may no longer hold. The obvious solution to this problem would be to redo the security analysis for those two protocols combined. Ideally, cryptographic protocol design should be modular, and a protocol should be built by composing other cryptographic primitives and protocols as subroutines. Modular design facilitates security analysis, but shared state remains an issue. Some composable models describe sufficient conditions for protocols to be securely composed even in the presence of shared state [41, 39]. It remains to see whether these general composability results will apply to the TPM, or whether novel, more precise notions will be needed.

Multi-user setting and policies. The TPM security definition must take into account that a single TPM can be used by multiple users concurrently. Furthermore, there may be different categories of users regarding how they can use the TPM. In general, each user may have its own authorization policy towards the TPM. Therefore, the security definition will need to incorporate a family of usage policies.

This list of challenges is not complete. It is difficult to foresee in advance all the possible obstacles that the FutureTPM project will encounter when defining and analyzing the security of TPMs. Addressing these challenges will require in part the use of existing techniques, but it could also need the extension or enhancement of existing security models, particularly for the case of protocols that share state.

6.3 Other Research Questions

In addition to the main research challenges discussed in Section 6.2, the following research problems could also be explored: although answering them is not critical to the successful development of a security model for the TPM, they would provide robust foundations for future developments of a similar nature, and support–through a deeper understanding of security for trusted computing in the presence of quantum-capable adversaries–a more sustainable way of developing security standards.

6.3.1 Quantum UC

As noted in Section 4.2.3, the question of whether the composition theorems provided by composable models still hold in the presence of a quantum adversary remains open. A natural and

very ambitious research direction would be to give an answer to this question, by providing a composable model that considers quantum adversaries with a composition theorem similar to the ones provided by composable frameworks that consider classical adversaries. Such a composable model would benefit research on quantum-resistant cryptographic protocols in general, not only quantum-resistant TPMs.
We remark that this could also impact the level of quantum security that our FutureTPM is proven to achieve. As can be seen, even if each of the cryptographic primitives implemented in the FutureTPM are proven to attain superposition-based quantum security (QS2, as defined in Section 2.3 of deliverable 2.1), if our TPM security definition is not in a security model that considers the adversaries used in superposition-based quantum security definitions, then our proof that the FutureTPM fulfils our TPM security definition will not be a proof that our TPM provides superposition-based quantum security.

6.3.2 Tool support for mechanized UC proofs

Although some UC proofs have been formalized (for example, by Haagh et al. [57]), this was achieved by first proving, with pen-and-paper, a reduction stating that a simulation-based result was sufficient.
In general, general proof assistants–or even those specialized to cryptography such as EasyCrypt or FCF–should be able to capture UC notions, at a significant cost in proof complexity. At present, there is no tool for mechanizing cryptographic security proofs that supports UC natively.
Developing such a tool would be a valuable contribution in itself, and could greatly contribute to the success of the analysis of a significant part of the FutureTPM design. However, another viable alternative would be to encode details of UC frameworks in such proof assistants in a systematic way–which would enable the mechanized proving of generic reductions such as the one used by Haagh et al.

6.3.3 Security models with side-channels

Leakage-resilient cryptography is an active research area that aims at bringing the abstract models and definitions that are commonly used to define security of cryptographic tasks closer to the real world where the cryptographic algorithms and protocols are run by capturing side-channels in security definitions.
In many FutureTPM deployments, considering attackers that are able to perform side-channel attacks on a TPM is reasonable. Therefore, providing a leakage-resilient definition of security for TPMs is an interesting goal. This security definition could consider leakage-resilience for one or more of the TPM functionalities. To realize such a definition, the FutureTPM will require cryptographic primitives and protocols that are both quantum-resistant and leakage-resilient for those functionalities.

6.3.4 Minimal hardware support for advanced trusted computing functionalities

The TPM–and the FutureTPM–do not support secure execution in general. ARM TrustZone, Intel SGX and Sanctum are architectures that support secure execution, but are also much more complex, and suffer from a much thinner interface between the adversary and the secure processor–with the same chip and shared resources being used for the execution of trusted and untrusted code.

An interesting research question would then be to identify a minimal set of functionalities that would enable the TPM to support secure execution of arbitrary code–in the same sense as realized by Intel SGX, but with much stronger isolation guarantees.

Chapter 7

List of Abbreviations

Abbreviation	Translation
AIK	Attestation Identity Key
CAPEC	Common Attack Pattern Enumeration and Classification
CRS	Common Reference String
CSP	Communicating Sequential Processes
CWE	Common Weakness and Enumeration
DAA	Direct Anonymous Attestation
DFD	Data Flow Diagram
EA	Enhanced Authorization
ECC	Elliptic Curve Cryptography
EK	Endorsement Key
KDF	Key Derivation Function
MVP	Minimum Viable Product
NV	Non-Volatile
NVRAM	Non-Volatile Random-Access Memory
OFMC	On-the-fly model checker
OWASP	Open Web Application Security Project
PCR	Platform Configuration Registers
QR	Quantum-Resistant
QSH	Quantum-Safe Hybrid
RM	Resource Manager
RNG	Random Number Generator
TAB	TPM Access Broker
TAP	Trusted Abstract Platform
TCTI	TPM Command Transmission Interface
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
TSS	Trusted Software Stack
UC	Universal Composability
VM	Virtual Machine

References

[1] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, July 2002.

[2] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and François Dupressoir. Verifiable side-channel security of cryptographic implementations: Constant-time MEE-CBC. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 163–184, 2016.

[3] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. A fast and verified software stack for secure function evaluation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1989–2006, 2017.

[4] Megumi Ando, Joshua D Guttman, Alberto R Papaleo, and John Scire. Hash-based TPM signatures for the quantum world. In *International Conference on Applied Cryptography and Network Security*, volume 9696 of *LNCS*, pages 77–94, Guildford, UK, June 2016. Springer.

[5] Myrto Arapinis, Joshua Phillips, Eike Ritter, and Mark D Ryan. Statverif: Verification of stateful processes. *Journal of Computer Security*, 22(5):743–821, July 2014.

[6] Alessandro Armando, Roberto Carbone, and Luca Compagna. SATMC: A SAT-based model checker for security-critical systems. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 8413 of *LNCS*, pages 31–45, Grenoble, France, April 2014. Springer.

[7] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 202–215, 2008.

[8] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129, 2016.

[9] Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech, and Benedikt Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. *IACR Cryptology ePrint Archive*, 2015:74, 2015.

[10] Gilles Barthe, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Making RSA-PSS provably secure against non-random faults. *IACR Cryptology ePrint Archive*, 2014:252, 2014.

[11] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, pages 146–166, 2013.

[12] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 71–90, 2011.

[13] Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago Zanella Béguelin. Beyond provable security verifiable IND-CCA security of OAEP. In *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, pages 180–196, 2011.

[14] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 90–101, 2009.

[15] Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. A machine-checked formalization of sigma-protocols. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 246–260, 2010.

[16] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, and Benedikt Schmidt Ralf Sasse. Tamarin prover. https://tamarin-prover.github.io.

[17] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, and Benedikt Schmidt Ralf Sasse. Tamarin prover. http://maude.cs.illinois.edu.

[18] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, June 2005.

[19] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. Crypthol: Game-based proofs in higher-order logic. *IACR Cryptology ePrint Archive*, 2017:753, 2017.

[20] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[21] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 409–426, 2006.

[22] David Bernhard, Georg Fuchsbauer, Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. Anonymous attestation with user-controlled linkability. *Int. J. Inf. Sec.*, 12(3):219–249, 2013.

[23] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.

[24] Bruno Blanchet. Composition theorems for cryptoverif and application to TLS 1.3. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 16–30, 2018.

[25] Bruno Blanchet, Vincent Cheval, and Marc Sylvestre. Proverif. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/.

[26] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 537–554, 2006.

[27] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. *IACR Cryptology ePrint Archive*, 2015:59, 2015.

[28] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

[29] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, pages 132–145, 2004.

[30] Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. *Int. J. Inf. Sec.*, 8(5):315–330, 2009.

[31] Chris Brzuska, Antoine Delignat-Lavaud, Konrad Kohbrok, and Markulf Kohlweiss. State-separating proofs: A reduction methodology for real-world protocols. *IACR Cryptology ePrint Archive*, 2018:306, 2018.

[32] David Cadé and Bruno Blanchet. From computationally-proved protocol specifications to implementations and application to SSH. *JoWUA*, 4(1):4–31, 2013.

[33] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 901–920, 2017.

[34] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 280–312, 2018.

[35] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, pages 234–264, 2016.

[36] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation with subverted tpms. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 427–461, 2017.

[37] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. Universal composition with responsive environments. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, pages 807–840, 2016.

[38] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

[39] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 61–85, 2007.

[40] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 597–608, 2014.

[41] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Annual International Cryptology Conference*, pages 265–281. Springer, 2003.

[42] Liqun Chen and Mark Ryan. Attack, solution and verification for shared authorisation data in TCG TPM. In *International Workshop on Formal Aspects in Security and Trust*, volume 5983 of *LNCS*, pages 201–216, Eindhoven, The Netherlands, November 2009. Springer.

[43] Web Application Security Consortium. Wasc threat classification v2.0. [Accessed on 20 July 2018].

[44] Véronique Cortier. Verification of security protocols. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 5403 of *LNCS*, pages 5–13, Savannah, GA, USA, January 2009. Springer.

[45] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 993–1008, 2017.

[46] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: Privacy and verifiability for belenios. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 298–312, 2018.

[47] Véronique Cortier and Steve Kremer. Formal models and techniques for analyzing security protocols: A tutorial. *Foundations and Trends in Programming Languages*, 1(3):151–267, 2014.

[48] Cas Cremers. The scyther tool. https://www.cs.ox.ac.uk/people/cas.cremers/scyther/.

[49] Jean Paul Degabriele, Kenneth G. Paterson, and Gaven J. Watson. Provable security in the real world. *IEEE Security & Privacy*, 9(3):33–41, 2011.

[50] Stéphanie Delaune, Steve Kremer, Mark D Ryan, and Graham Steel. Formal analysis of protocols based on TPM state registers. In *24th IEEE Computer Security Foundations Symposium (CSF 2011)*, pages 66–80, Cernay-la-Ville, France, June 2011. IEEE.

[51] Stéphanie Delaune, Steve Kremer, Mark D Ryan, and Graham Steel. A formal analysis of authentication in the TPM. In *International Workshop on Formal Aspects in Security and Trust*, volume 6561 of *LNCS*, pages 111–125, Pisa, Italy, September 2010. Springer.

[52] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 463–482, 2017.

[53] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983.

[54] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[55] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[56] Sigrid Gürgens, Carsten Rudolph, Dirk Scheuermann, Marion Atts, and Rainer Plaga. Security evaluation of scenarios based on the TCG's TPM specification. In *European Symposium on Research in Computer Security*, volume 4734 of *LNCS*, pages 438–453, Dresden, Germany, September 2007. Springer.

[57] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. Computer-aided proofs for multiparty computation with active security. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 119–131, 2018.

[58] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. Uncover security design flaws using the stride approach. MSDN Magazine, 2006. [Accessed on 20 July 2018].

[59] Dennis Hofheinz and Victor Shoup. Gnuc: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, 2015.

[60] S. Hussain, G. Rasool A. Kamal S. Ahmad, and S. Iqbal. *Threat modelling methodologies: A survey*. Sci. Int.(Lahore), 2014.

[61] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.

[62] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[63] S. Krishnan. A hybrid approach to threat modelling. 2017.

[64] Mitre. Capec home page. [Accessed on 20 July 2018].

[65] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[66] University of Oxford. Fdr4 - the CSP refinement checker. https://www.cs.ox.ac.uk/projects/fdr/.

[67] OWASP. Owasp home page. [Accessed on 20 July 2018].

[68] Rafael Pass, Elaine Shi, and Florian Tramer. Formal abstractions for attested execution secure processors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 10210 of *LNCS*, pages 260–289, Paris, France, April 2017. Springer.

[69] Kenneth G. Paterson and Gaven J. Watson. Plaintext-dependent decryption: A formal security treatment of SSH-CTR. *IACR Cryptology ePrint Archive*, 2010:95, 2010.

[70] Adam Petcher and Greg Morrisett. The foundational cryptography framework. In *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, pages 53–72, 2015.

[71] The Maude Project and Team. Maude-npa. http://maude.cs.illinois.edu/w/index.php?title=Maude_Tools:_Maude-NPA.

[72] Phillip Rogaway. Practice-oriented provable security and the social construction of cryptography. *IEEE Security & Privacy*, 14(6):10–17, 2016.

[73] Mark Ryan, Myrto Arapinis, and Eike Ritter. Statverif. https://sec.cs.bham.ac.uk/research/StatVerif/.

[74] Mark D Ryan. Automatic analysis of security properties of the TPM. In *International Conference on Trusted Systems*, volume 7711 of *LNCS*, pages 1–4, London, UK, December 2012. Springer.

[75] Jianxiong Shao, Dengguo Feng, and Yu Qin. Type-based analysis of protected storage in the TPM. In *International Conference on Information and Communications Security*, volume 8233 of *LNCS*, pages 135–150, Beijing, China, November 2013. Springer.

[76] Jianxiong Shao, Yu Qin, Dengguo Feng, and Weijin Wang. Formal analysis of enhanced authorization in the TPM 2.0. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 273–284, Singapore, April 2015. ACM New York.

[77] A Shostack. *Threat Modelling: Designing for Security*, volume 26. Indianapolis, USA: Wiley, 2014.

[78] Ben Smyth, Mark Ryan, and Liqun Chen. Formal analysis of anonymity in ECC-based Direct Anonymous Attestation schemes. In *International Workshop on Formal Aspects in Security and Trust*, volume 7140 of *LNCS*, pages 245–262, Leuven, Belgium, September 2011. Springer.

[79] Ben Smyth, Mark D Ryan, and Liqun Chen. Formal analysis of privacy in Direct Anonymous Attestation schemes. *Science of Computer Programming*, 111:300–317, 2015.

[80] Pramod Subramanyan, Rohit Sinha, Ilia Lebedev, Srinivas Devadas, and Sanjit A Seshia. A formal foundation for secure remote execution of enclaves. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2435–2450, Dallas, Texas, USA, November 2017. ACM New York.

[81] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. In *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 266–278, 2011.

[82] TCG. Trusted Platform Module library part 1: Architecture. Technical report, Trusted Computing Group, 2016. https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf.

[83] Loria-INRIA CASSIS Team. Cl-atse: Constraint logic-based attack searcher. https://cassis.loria.fr/wiki/Wiki.jsp?page=Cl-Atse.

[84] Fondazione Bruno Kessler Security & Trust. Satmc: Sat-based model-checker for security protocols and security-sensitive applications. https://st.fbk.eu/technologies/satmc-tool.

[85] Technical University of Munich et al. University of Cambridge. Isabelle proof assistant. https://isabelle.in.tum.de/index.html.

[86] Dominique Unruh. Universally composable quantum multi-party computation. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 486–505, 2010.

[87] T.U. Velez and M.M. Morana. *Risk Centric Threat Modelling. Process for Attack Simulation and Threat Analysis.* New Jersey: Wiley, 2015.

[88] Weijin Wang, Yu Qin, and Dengguo Feng. Automated proof for authorization protocols of TPM 2.0 in computational model. In *International Conference on Information Security Practice and Experience*, volume 8434 of *LNCS*, pages 144–158, Fuzhou, China, May 2014. Springer.

[89] Weijin Wang, Yu Qin, Bo Yang, Yingjun Zhang, and Dengguo Feng. Automated security proof of cryptographic support commands in TPM 2.0. In *International Conference on Information and Communications Security*, volume 9977 of *LNCS*, pages 431–441, Singapore, November 2016. Springer.

[90] Li Xi and Dengguo Feng. Formal analysis of DAA-related APIs in TPM 2.0. In *International Conference on Network and System Security*, volume 8792 of *LNCS*, pages 421–434, Xi'an, China, October 2014. Springer.

[91] Jiun Yi Yap and Allan Tomlinson. Threat model of a scenario based on trusted platform module 2.0 specification. In *WASH*, 2013.

[92] Fajiang Yu, Huanguo Zhang, Bo Zhao, Juan Wang, Liqiang Zhang, Fei Yan, and Zhenlin Chen. A formal analysis of Trusted Platform Module 2.0 hash-based message authentication code authorization under digital rights management scenario. *Security and Communication Networks*, 9(15):2802–2815, 2016.

[93] Santiago Zanella-Béguelin, Gilles Barthe, Benjamin Grégoire, and Federico Olmedo. Formally certifying the security of digital signature schemes. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 237–250, 2009.

[94] Qianying Zhang, Shijun Zhao, Yu Qin, and Dengguo Feng. Formal analysis of TPM2.0 key management APIs. *Chinese Science Bulletin*, 59(32):4210–4224, August 2014.