Formal Analysis and Implementation of a TPM 2.0-based Direct Anonymous Attestation Scheme

Stephan Wesemeyer
s.wesemeyer@surrey.ac.uk
University of Surrey

Christopher J.P. Newton
c.newton@surrey.ac.uk
University of Surrey

Helen Treharne
h.treharne@surrey.ac.uk
University of Surrey

Liqun Chen
liqun.chen@surrey.ac.uk
University of Surrey

Ralf Sasse
ralf.sasse@inf.ethz.ch
ETH Zurich

Jorden Whitefield
jorden.whitefield@ericsson.com
Ericsson AB, Finland

ABSTRACT

Direct Anonymous Attestation (DAA) is a set of cryptographic schemes used to create anonymous digital signatures. To provide additional assurance, DAA schemes can utilise a Trusted Platform Module (TPM) that is a tamper-resistant hardware device embedded in a computing platform and which provides cryptographic primitives and secure storage. We extend Chen and Li's DAA scheme to support: 1) signing a message anonymously, 2) self-certifying TPM keys, and 3) ascertaining a platform's state as recorded by the TPM's platform configuration registers (PCR) for remote attestation, with explicit reference to TPM 2.0 API calls. We perform a formal analysis of the scheme and are the first symbolic models to explicitly include the low-level TPM call details. Our analysis reveals that a fix proposed by Whitefield *et al.* to address an authentication attack on an ECC-DAA scheme is also required by our scheme. Developing a fine-grained, formal model of a DAA scheme contributes to the growing body of work demonstrating the use of formal tools in supporting security analyses of cryptographic protocols. We additionally provide and benchmark an open-source C++ implementation of this DAA scheme supporting both a hardware and a software TPM and measure its performance.

KEYWORDS

Formal Verification, Security Protocols, TPM, DAA

ACM Reference Format:
Stephan Wesemeyer, Christopher J.P. Newton, Helen Treharne, Liqun Chen, Ralf Sasse, and Jorden Whitefield. 2019. Formal Analysis and Implementation of a TPM 2.0-based Direct Anonymous Attestation Scheme. In *Proceedings of ACM Conference (ASIACCS '20).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Devices such as laptops, smartphones, tablets, and Internet of Things devices, which connect to the Internet, are commonplace. However, establishing their current state in terms of their configuration, security settings and "trustworthiness" is difficult. Trusted computing is one approach that enhances the security on these devices by installing a "root of trust" (RoT). These roots of trust can be used to attest that devices are in a "trustworthy" state, meaning that the devices should behave as expected for a specific purpose.

The Trusted Platform Module (TPM) is a separate chip especially designed to be such a RoT and the Trusted Computing Group (TCG) reports there are billions of TPMs embedded in branded PCs, laptops and servers [44]. A TPM is a resource-constrained cryptographic co-processor that is embedded within a commodity device which we refer to as the HOST. The combination of a HOST and TPM is called a PLATFORM. Software can execute cryptographic operations on the TPM via its Application Programming Interface (API). A TPM can, when required, store cryptographic keys and other sensitive data in protected memory. Note, that a RoT must be guaranteed by a trusted third party, *e.g.*, a TPM manufacturer such as Infineon, before it can act as a RoT for a PLATFORM.

TPMs can be used to collect evidence and provide signed reports about a HOST's code and data during boot time, *e.g.*, the time it takes to load code from disk to memory or compute hashes of installed software to detect file-tampering.

The TCG have defined and standardised the TPM commands [38] necessary to provide a range of services, including: attesting PLATFORM configuration registers (PCR) and attesting other TPM objects, *e.g.*, keys, clock, time and audit digest data. These attestation commands enable a TPM to sign an internally generated data structure. A TPM can also be used to sign an externally provided digest. All of these signing commands can either be performed anonymously or otherwise. Quoting PCR values is particularly important in order to verify the trustworthiness of a PLATFORM.

For some use cases, it is desirable that these signing operations can be conducted using a secure and privacy-preserving scheme which protects users' privacy, and reduces the knowledge that external entities can learn about a system. Moreover, the level of protection should be under the user's control. One scheme that provides such protection is Direct Anonymous Attestation (DAA) [6]. DAA is an anonymous signature scheme designed to provide anonymous digital signatures which can be used for device attestation and signing while providing user-controlled anonymity and privacy. The TCG have defined the commands in TPM 2.0 to flexibly support a number of different DAA schemes.

One such scheme is an Elliptic-Curve-DAA scheme (ECC-DAA) which comprises five main operations: SETUP, JOIN, SIGN, VERIFY,

ASIACCS '20, June 2020, Taipei, Taiwan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and Link. The Setup and Join are used to guarantee the Tpm as the RoT with the help of a trusted third party. Sign, Verify and Link are respectively used to sign messages or attestation data anonymously, verify those signatures and allow a user to link two or more such anonymous signatures if desired.

Chen *et al.* [16] defined an Ecc-Daa scheme in terms of Tpm commands and described, at a high level, the Setup, Join, Sign and Verify phases of their scheme. They also provided a computational security proof of the Tpm command TPM2_Sign that is instrumental in both the Join and Sign phases and on which the TPM2_Sign command in the TCG documentation was based. Their scheme, however, does not provide sufficient detail for an implementation.

We address this lack of detail by defining the implementation-level details for Chen *et al.* 's Ecc-Daa scheme [16] using the most recent Tpm hardware version: Tpm 2.0, version 1.38 (a Tpm evaluation module made by Infineon for the Raspberry Pi [25]). Using the available Tpm 2.0 commands, we explicitly specify the sequence of Tpm 2.0 calls needed to implement the various Ecc-Daa phases. Moreover, apart from supporting signing a message anonymously we extend the Sign phase of the scheme to support the attestation (self-certifying) of Tpm keys and the attestation (quoting) of PCR values.

Contributions: We provide three main contributions. The first contribution is the Tpm 2.0 API level detail of Chen *et al.* 's scheme [16] using the most recent Tpm 2.0 specification and its extension to include Certify and Quote operations. Second, we provide a formal analysis of the scheme using the Tamarin Prover. The models used for the analysis were specifically designed to include Tamarin representations of the low-level Tpm 2.0 calls involved in the scheme. We state the authentication and security properties required by the scheme based on Chen [14] and prove these symbolically using the Tamarin Prover [35], which is a symbolic modelling and analysis tool. Our analysis is based on the approach taken by Whitefield *et al.* [46] and we verify that their proposed fix for an authentication attack is also required in our extended version of Chen *et al.* 's scheme. Moreover, we correct Whitefield *et al.* 's definition of user-controlled anonymity. Third, we make an open-source reference implementation of the scheme available in C++ and benchmark its performance on a Raspberry Pi (using an ARMv7 processor and Infineon's hardware Tpm). Our implementation can be used as a basis for further development of Ecc-Daa schemes using Tpm 2.0 and comparative benchmarking. It should also lower the barrier of entry for other researchers who want to explore Tpm-based solutions. Note that our Ecc-Daa implementation also runs on an Intel x86 platform using a software Tpm emulator [26].

All the Tamarin models and the C++ reference implementation can be found at [18, 19].

2 RELATED WORKS

Brickell, Camenisch and Chen originally proposed Daa [6] and used a simulation based proof to verify their scheme. Brickell, Chen and Li [7, 8] subsequently proposed an Ecc-Daa scheme based on elliptic curve pairings and introduced a game based model to prove their scheme. Their schemes identified security notions that included correctness, user-controlled anonymity, user-controlled traceability, and we introduce these properties in detail in Section 3.2. Chen's scheme [14] improved upon the efficiency of the Brickell, Chen and Li [8] scheme so that fewer Tpm calls were required, and non-frameability was introduced as a further security property to be considered. Chen, Morrissey and Smart [17] proposed a more efficient Ecc-Daa scheme than Brickell, Chen and Li [8]'s scheme by using Type III pairings. Chen *et al.* [21] proposed an implementation of an improved version of the Chen, Morrissey and Smart [17] scheme. This implementation was developed entirely in software and covered all of the phases of an Ecc-Daa scheme. Nonetheless, no documentation of the Tpm calls required to implement the Ecc-Daa scheme was included in the implementation and the source code is not provided.

Recently, Yang *et al.* [47] introduced an implementation of an Ecc-Daa scheme based on Tpm 2.0 calls. They particularly focus on the Sign operation and reduce the number of exponentiations required. They provided a rigorous analysis and an associated implementation of the Sign operation.

Another family of Ecc-Daa schemes is Enhanced Privacy ID (EPID). Brickell and Li proposed an EPID scheme [10] and proved it in a game based model. Luther [34] provides implementations of two Daa schemes: EPID [10], and the Lightweight Anonymous Attestation Scheme with Efficient Revocation [31] on Tpm 2.0. Acar *et al.* [1] demonstrated that the API for a Tpm allows an adversary to use a Tpm as a static Diffie-Hellman (DH) oracle. Brown and Gallant [11] found that although solving a static DH problem is still computationally infeasible, it is simpler to solve than the computational DH problem.

Whitefield *et al.* [46] performed a detailed symbolic analysis of the ISO/IEC 20008-2 standardised Daa scheme based on elliptic curves (Ecc-Daa) in which they found and provided a fixed for an authentication attack. While they also formally verified their Ecc-Daa scheme, their analysis is not as fine-grained as ours and their definition of user-controlled anonymity is too strong.

Camenisch, Drijvers and Lehman [13] have proposed a UC model for an Ecc-Daa scheme. Their scheme preserves a new set of security properties. Subsequently, Camenisch *et al.* [12] added an additional property that should be preserved in the presence of malicious parties, and their security analysis is also based on UC models.

As our interest is in defining Ecc-Daa schemes which use Tpms, we used Chen *et al.* 's scheme [16] as the basis since this is one of the most recent schemes that focuses on detailed Tpm 2.0 calls. We defined the security properties of the scheme based on the work by Chen [14] rather than those presented more recently in Camenisch *et al.* [12, 13] as their scheme cannot yet be implemented by the current Tpm 2.0 specification.

3 DIRECT ANONYMOUS ATTESTATION

Daa is a special group signature scheme where the group manager (Issuer) issues credentials which can be used to verify that a signature was created by a valid member of the group. However, unlike other group signature schemes, given a valid signature the group manager is unable to identify the signer.

Chen *et al.* defined a Ecc-Daa scheme [16] with five operations, Setup, Join, Sign, Verify and Link as summarised below. This paper makes explicit that the Sign operation can cover three distinct contexts.

- Setup - system parameters must be chosen and the Issuer needs to generate its keys. The system parameters and the Issuer's public keys are then published and available to the group and to anyone who needs to verify the validity of a signature.
- Join - a Host using a Tpm joins the group and obtains an attestation key (AK) credential for an Ecc-Daa key created by the Tpm. The key can then be used to anonymously sign a message, or attest to data from this Tpm.
- Sign - this operation uses an Ecc-Daa key to sign data in three different contexts: to sign a message (Sign), to attest another key generated by the Tpm (Certify) and to attest PCR values confirming the state of the system (Quote).
- Verify - verifying a signature and returning true (valid) or false (invalid).
- Link - checking two signatures to see if they are linked and returning true (linked) or false (un-linked). Signatures using the same Ecc-Daa key can be linked by using the same basename in the signature process.

3.1 Notation

The following symbols and abbreviations apply throughout the paper:

t	A security parameter.
p, n	Prime numbers.
$[x, y]$	The set of integers from x to y inclusive, if x, y are integers satisfying $x \leq y$.
$\mathbb{Z}_p$	The set of integers modulo p, *i.e.*, $[0, p-1]$. These form a prime field, $\mathbb{F}_p$. Also note that $\mathbb{F}_{p^m}$ is an extension finite field with p^m elements, where m is a positive integer.
$\mathbb{Z}_p*$	The multiplicative group of invertible elements in $\mathbb{Z}_p$, *i.e.*, the set of integers in $[1, p-1]$.
$\mathbb{G}_1$	An additive cyclic group of order n over an elliptic curve. The curve has points with co-ordinates in $\mathbb{F}_p \times \mathbb{F}_p$.
P_1	A generator of $\mathbb{G}_1$.
$\mathbb{G}_2$	An additive cyclic group of order n over an elliptic curve. The curve has points with co-ordinates in $\mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$.
P_2	A generator of $\mathbb{G}_2$.
$[k]P$	Multiplication operation that takes a positive integer k and a point P on the elliptic curve E as input and produces as output another point Q on the curve E, where $Q = [k]P = P + P + \ldots + P$, *i.e.*, the sum of k copies of P.
$\mathbb{G}_T$	A multiplicative cyclic group of order n.
$\hat{h}$	A bilinear map $\widehat{h} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that for all $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$, and all positive integers a, b, the equation $\hat{h}([a]P, [b]Q) = \hat{h}(P, Q)^{ab}$ holds. This bilinear map is also called a pairing function.
H	A cryptographic hash-function. We need several hash functions: - $H_p : \{0,1\}^* \rightarrow \mathbb{Z}_p$ is used when we want to hash to a co-ordinate on the elliptic curve, - $H_n : \{0,1\}^* \rightarrow \mathbb{Z}_n$ is used when we want to hash to a multiplier for an elliptic curve point and - $H_k : \{0,1\}^* \rightarrow \mathbb{Z}_{2^k}$ a general hashing function.
H_s	The 'map to point' function, used to map a random string, rs, to a tuple $H_s(rs) = (\bar{s}_2, y_2)$ such that $(H_p(\bar{s}_2), y_2)$ is a point on the curve $\mathbb{G}_1$. The function H_s is essentially the function I2P given in ISO/IEC standard document 11770-4 [28] using SHA256 as the Key Derivation Function (KDF).
$\perp$	The item is unspecified. To avoid unnecessary if–then–else constructs in the diagrams, this carries through. So, for group items, for example, if $A = \perp$, then $[b]A$ will also equal $\perp$.
$\text{len}_{16}(x)$	The length of x expressed as a 16-bit integer (most significant bit first).
senc	A symmetric-key encryption function, senc(*data*, *key*).
sdec	A symmetric-key decryption function, sdec(*cipher*, *key*).
aenc	An asymmetric encryption function, aenc(*data*, *ekey*).
adec	An asymmetric decryption function, adec(*cipher*, *dkey*).
m	A message to be signed.

3.2 Security Properties

In this section we provide an intuitive description of the security properties our scheme is designed to provide. These properties were first introduced by Brickell *et al.* [8] and then refined by Chen [14]. More recently, Camenisch *et al.* [13] have proposed an alternative set of security properties but their scheme cannot yet be implemented by the current Tpm 2.0 specification. In this paper, since our scheme is based on Chen *et al.* and their proofs, our definitions use the terminology presented in [14].

Recall that we use the term Platform to describe the unique, one-to-one combination of a Host and a Tpm. We use the term message to represent a signature, a certified key or a quoted PCR value that can be verified by a Verifier. Note that the Tpm component of a Platform is the trusted element that generates the signatures, certified keys and quoted PCR values in conjunction with the (potentially untrusted) Host.

SP1: Correctness: requires the scheme to be consistent and ensures that, in the absence of an adversary, the protocol can be executed correctly.

User-controlled traceability: this property covers the three notions of:

- **SP2: User-controlled linkability:** given a single basename $bsn(\neq \perp)$, an adversary finds it hard to create two different messages under the same private key with both messages associated with that *bsn*, but the messages are not linked.
- **SP3: Unforgeability:** an adversary, who is in the possession of a set of Platforms' private keys and associated credentials, finds it hard to forge a valid message for a private key and credential, which is not in that set.
- **SP4: Non-frameability:** no combination of dishonest Issuers and Platforms can create a valid message m_0 which can be linked to some given message m_1 generated by an honest Platform, unless that Platform produced the message m_0.

User-controlled anonymity: this property combines the two notions of:

- **SP5: Anonymity:** an adversary, who does not know the private key of a Platform, finds it hard to recover the identity of the Tpm used by the Platform from a given message.
- **SP6: User-controlled unlinkability:** given two messages m_0 and m_1 associated with two basenames bsn_0 and bsn_1 respectively, where $bsn_0 \neq bsn_1$, an adversary, who does not know the private key(s) of the associated Platforms, finds

it hard to tell whether or not the two messages originated from the same Platform.

We propose the following additional security property based on the work done by Whitefield *et al.* [46] which addresses a particular issue during the Join process:

SP7: Agreement during the Join: this property ensures that both the Platform and Issuer confirm each other's identity and agree on the credentials that were created during the Join operation.

Our work provides the missing fine-grained details of the scheme proposed by Chen *et al.* [16] which in turn is based on the work of [7–9, 14, 21]. This body of research has game based models for an Ecc-Daa scheme satisfying all the above properties (SP1-SP6). The security analysis for the whole scheme at the Tpm command level does not exist, apart from the detailed analysis of the signing phase using Tpm commands (TPM2_Commit and TPM2_Sign) [16]. The proofs of the schemes are independent of the Tpm commands, and provide the mathematical soundness of the overall schemes. Consequently, for our Tamarin analysis we do not require a game based proof of an Ecc-Daa scheme in terms of its Tpm commands since the assumptions we rely upon are at a mathematical level and not at the Tpm command level.

3.3 Tpm Key Management and Usage

In this section we explain how a Tpm generates and manages its keys, an understanding of which is needed in order to implement an Ecc-Daa scheme that utilises a Tpm.

One of the critical functions of a Tpm is to provide any keys that are needed by the Host. The Tpm's keys form a hierarchy with a primary key at the top and other child keys below. When a child key, K, is created, the Tpm passes back information to the Host which includes public data about the key K_{PD} and its sensitive data K_{SD}. K_{PD} contains the public key and information about how the key was created. K_{SD} contains the private key and is encrypted ("sealed") under its parent's AES key. However, when a primary key is created we only receive its public data from the Tpm. The private data can be stored in the Tpm's persistent memory, or re-created as needed. The keys are handled in this way as the Tpm has limited storage capacity and so, apart from any primary keys, the key data is stored outside of the Tpm. Keys need to be loaded before they can be used and flushed from the Tpm when not required to make space for other keys.

Note that there are several key hierarchies available in a Tpm [38], but in our implementation we only use the endorsement key hierarchy. This hierarchy requires external certification of the primary key which then allows it to be used to endorse other keys. This primary key in the endorsement hierarchy is called the endorsement key and denoted by $(e, \mathcal{E})$, where e is the private key and $\mathcal{E}$ is the corresponding public key. As this is a primary key, only its public data, $\mathcal{E}_{PD}$, which includes $\mathcal{E}$, is passed back to the host. All other keys in our implementation (*i.e.*, the Ecc-Daa key and any other key to be certified) are children of this endorsement key.

3.4 The Setup Operation

Before an Ecc-Daa scheme can be used, its parameters need to be agreed between all parties involved. Setting these parameters may be done by the Issuer or by some other authority as follows:

(1) Choose the security parameter t.
(2) Choose an asymmetric bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$ of large prime order n and an associated pairing function $\widehat{h}$.
(3) Choose two random generators P_1 and P_2.

Once this is done the Issuer takes the following steps:

(1) Chooses two random integers x, y in $\mathbb{Z}_n$. These are the Issuer's private keys.
(2) Computes $X = [x]P_2$ and $Y = [y]P_2$. These are the Issuer's public keys.
(3) Publishes the following information:
   – group public parameters = $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{h}, P_1, P_2, H_n, H_p, H_k)$,
   – group public key = (X, Y),

3.5 The Join Operation

The algorithm for the Join operation is shown in Figures 1 and 2. Note that information that is known to the different entities is given at the top of the protocol diagrams. Recall that during the Join operation the Host communicates with the Issuer to obtain an attestation key credential to use with its Ecc-Daa key. Calls to the Tpm are used to obtain the necessary keys and information that the Issuer requires. A number of important points are now made about this protocol. The letters corresponding to the items marked on the relevant protocol diagrams.
For Figure 1:

(a) The Tpm creates an Ecc-Daa key whose public key is Q_s. The host receives both the public data Q_{PD} and the sensitive data Q_{SD} of this Ecc-Daa key. The hosts then sends the Q_{PD} and the public endorsement key, $\mathcal{E}$ to the issuer. At an implementation level, this key must be: fixed to this Tpm, fixed to its parent in the Tpm's key hierarchy and restricted so that it will only sign digests that it has generated itself.
(b) The issuer creates a challenge $\mathcal{K}_1$ and encapsulates it using the make credential procedure. The result of this procedure is passed to the Platform which uses the TPM2_ActivateCredential call to unpack the data and retrieves $\mathcal{K}_1$. Implementation level details of both of these credential procedures are detailed in Appendix A.1.
(c) Notably, the construction of str includes the public endorsement key, $\mathcal{E}$. This addition was identified as an important requirement to protect against attacks with a rogue Tpm by Whitefield *et al.* [46]. The next three steps create a signature for str using three distinct Tpm calls.
(d) The inputs for this call to TPM2_Commit are all $\perp$, and in this case the commit command returns the point E.
(e) The data is hashed using TPM2_Hash which also returns a ticket, tkt, that is used to confirm that the hash was generated by the Tpm.
(f) Using TPM2_Sign the Tpm checks tkt and generates a signature.
(g) The Issuer confirms that the value of $\mathcal{K}_1$ returned by the Platform correspond to the original challenge and verifies the Host's signature, σ_{ch}.

At this point in the Join operation, the Issuer is convinced that the Ecc-Daa and endorsement keys both belong to the same Tpm

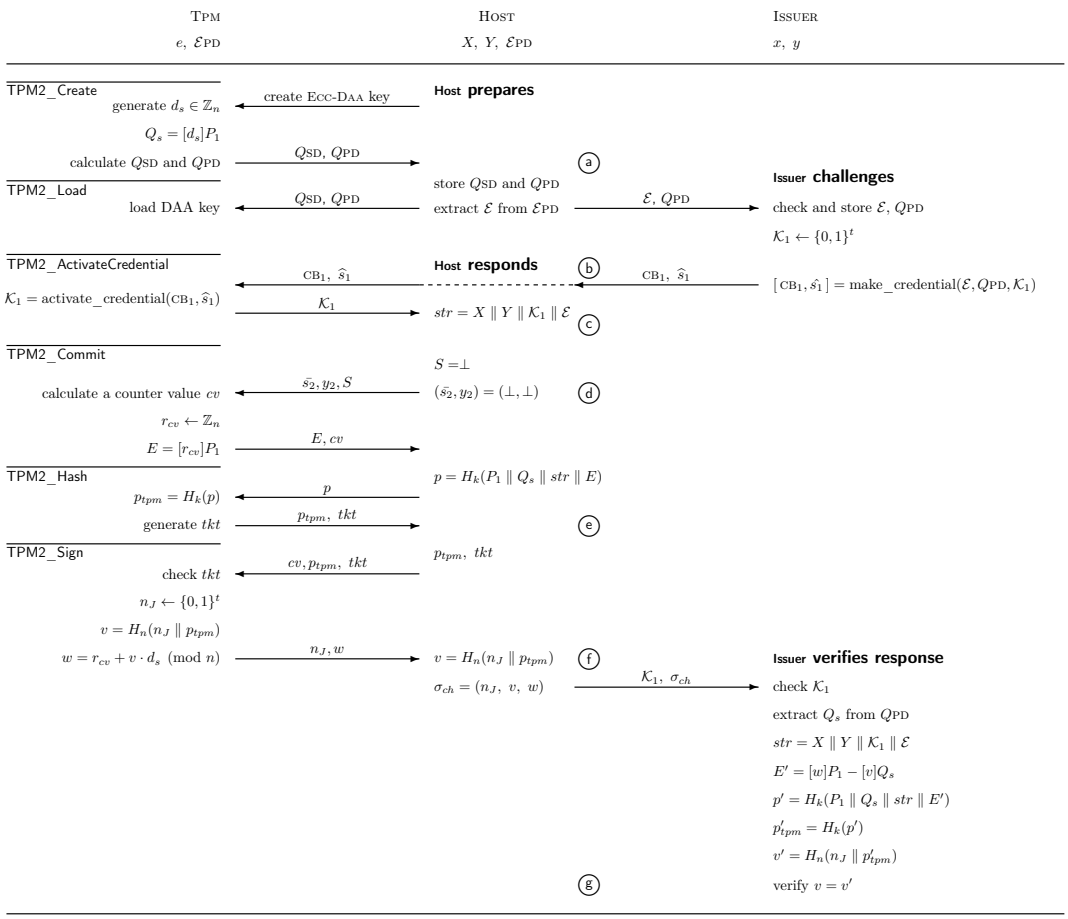

Figure 1: Initiating the JOIN

and that the ECC-DAA key that it received can be used for signing by the HOST. The remainder of the JOIN creates the attestation key credential, (A, B, C, D), for the ECC-DAA key as shown in Figure 2. For Figure 2:

(a) The ISSUER calculates the attestation key credential, (A, B, C, D), for the ECC-DAA key.
(b) The ISSUER signs this credential, σ_{cre}. This is effectively a double Schnorr signature that confirms to the host that B and D are correctly related, *i.e.*, B and P_1 and D and Q_s have the same discrete logarithm.
(c) The ISSUER creates an encryption key, $\mathcal{K}_2$ and uses it to encrypt the attestation key credential. The ISSUER encapsulates $\mathcal{K}_2$ using the make credential procedure. The result of the procedure together with the encrypted attestation key credential are sent to the HOST.
(d) The HOST uses TPM2_ActivateCredential to obtain the key, $\mathcal{K}_2$. This key is then used to decrypt $\widehat{C}$ and obtain the ECC-DAA key's credential.
(e) The HOST then checks the ISSUER's signature.
(f) The HOST confirms that the attestation key credential is valid using the bilinear map, $\hat{h}$.

3.6 The SIGN Operation

Signing with an ECC-DAA key is a two stage process. First, the attestation key credential (A, B, C, D) is randomised and the TPM2_Commit command is used to prepare parameters for the subsequent signing process. Second, the ECC-DAA key is used for any of the three distinct signing contexts (SIGN, CERTIFY and QUOTE).

Figure 3 shows the first stage, assuming that the required keys have been loaded into the TPM. The linking basename, *bsn*, is either unset ($\perp$) or is set to $\{0,1\}^*$. Note that if $bsn = \perp$, then $\bar{s}_2$ and y_2 will both be $\perp$ and remembering that we have said that $\perp$ carries through in any formula we see that TPM2_Commit just returns $E = [r_{cv1}]S$. K and L will both be unset as will J (all will be $\perp$). These then carry through for all of the other calculations.

Signatures using the same basename ($\neq\perp$) will have the same values of J and K which tells a VERIFIER that they were generated by the same signer and can therefore be linked. Nevertheless, the VERIFIER cannot tell the identity of the signer from this and the signer's anonymity is preserved.

Figures 4, 5 and 6 show the second stage. The mechanism for signing a message, m, using TPM2_Sign is shown in Figure 4. In the figure:

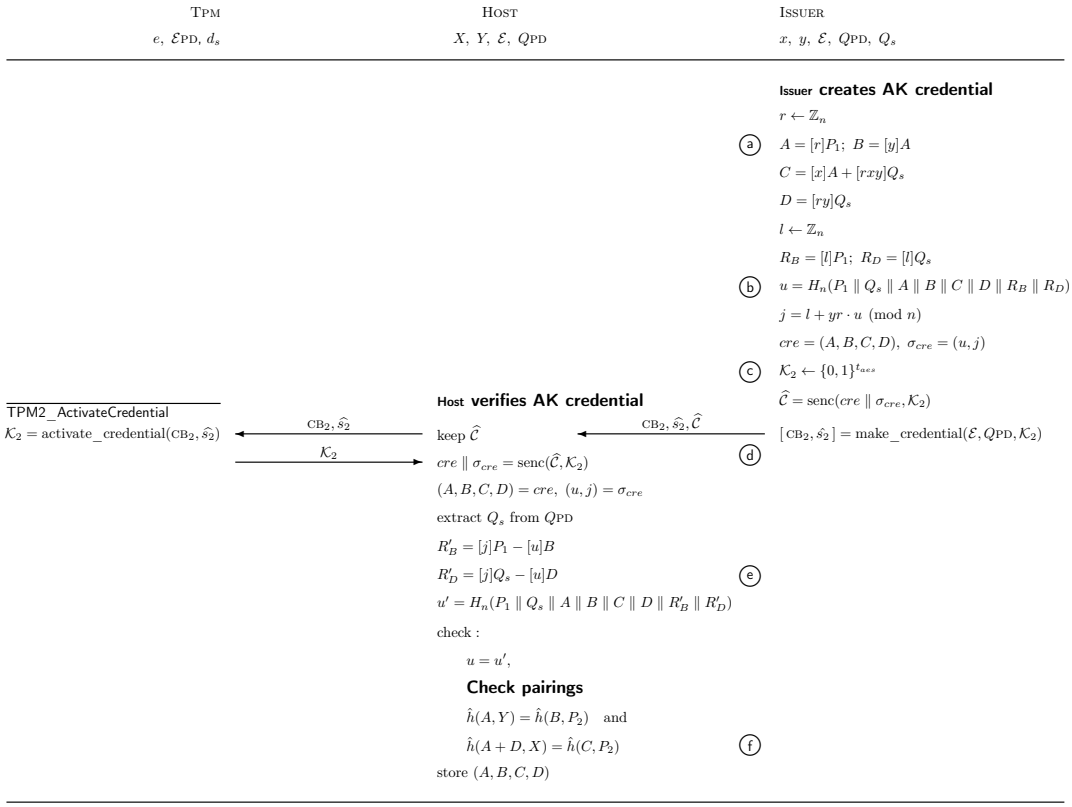

Figure 2: Completing the JOIN

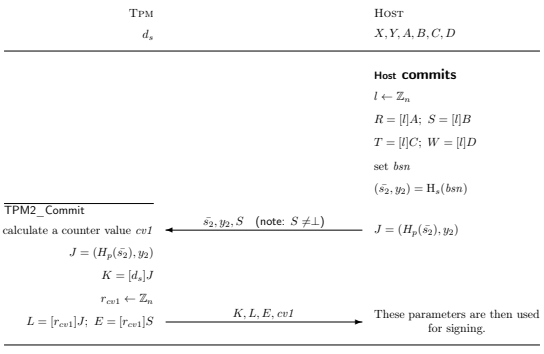

Figure 3: Preparing to use the ECC-DAA key

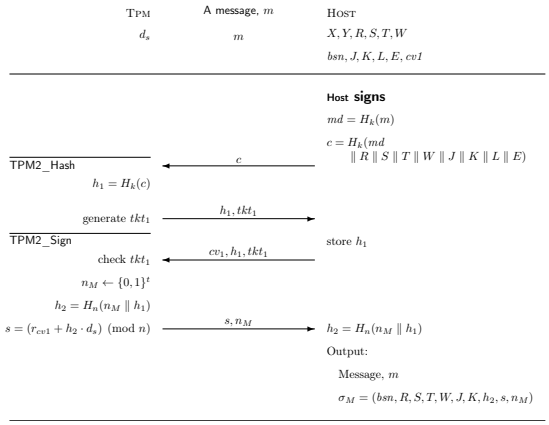

Figure 4: Signing a message, m (SIGN)

(a) If J, K and L are all $\perp$ they make no contribution to the hash, H_k.
(b) As before a ticket, tkt_1, is used to confirm that the digest, h_1, being signed has been generated by the TPM.

Figure 5 shows the mechanism for using TPM2_Certify to attest (self-certify) that a key Q_K was generated by and is loaded into the TPM. Figure 6 shows the mechanism for attesting to a set of PCR values (quoting) where TPM2_Quote is used to report on the state of the PLATFORM as recorded by its PCR values. As the data that they are signing is internally generated there is no need for the extra TPM2_Hash function. Information on the attestation data generated by these commands is given in Appendix A.2.

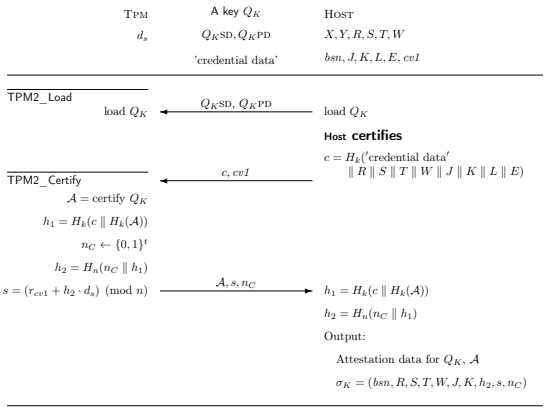

Figure 5: Certifying a key, Q_K (CERTIFY)

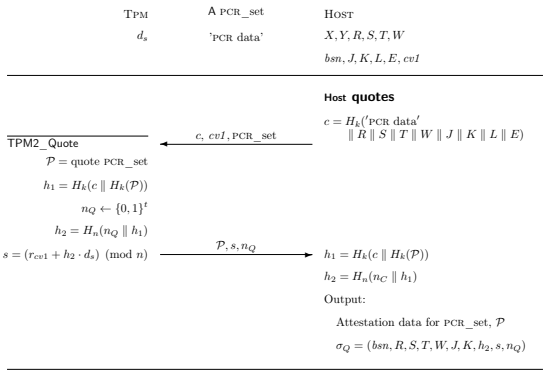

Figure 6: Quoting a set of PCR values (QUOTE)

3.7 The VERIFY and LINK Operations

Figure 7 shows the procedure for verifying the different ECC-DAA signatures. This consists of two parts: a) checking the ECC-DAA credential; and b) checking the signature itself.

Note that the VERIFIER needs access to the ISSUER's public keys, X and Y. This is shown at the top of Figure 7 together with the information needed for verifying the different signatures. Additionally, if $bsn = \perp$ then J, K and therefore L' will all be unset ($= \perp$) and will make no contribution to hash c'.

The VERIFIER can also check that two signatures are linked, if their corresponding J and K values are equal and not $\perp$ (not shown in Figure 7).

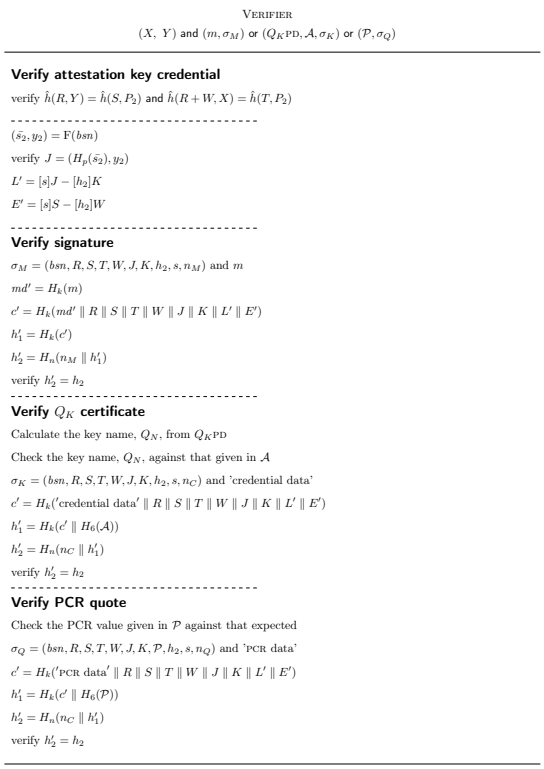

Figure 7: Verifying the ECC-DAA signatures (VERIFY)

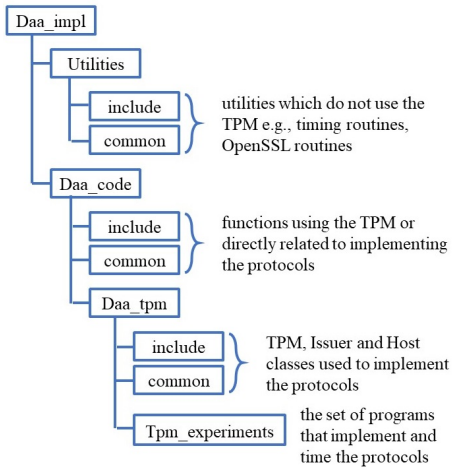

Figure 8: Structure of the C++codebase

4 IMPLEMENTATION

Implementation was straightforward once the protocols were written out in terms of the TPM calls (see Appendix A.3). The protocols were implemented in C++, using the GNU GCC compiler [23], the IBM implementation of a TPM software stack (IBM TSS) [27], OpenSSL [36] and the Apache Milagro crypto library (AMCL) [42]. There were approximately 7,900 lines of code (excluding all of the library code). The structure of the code is shown in Figure 8.

The protocols were tested on two platforms: a Raspberry Pi 3 (ARM v7) with an Infineon TPM [25] and an Intel x86_64 laptop with the TPM calls implemented using the IBM TPM emulator [26]. The protocols were benchmarked on both platforms and the results are presented in Appendix A.5.

The TPM specification includes two pairing friendly EC curves, BN_P256 and BN_P638. Recent developments in cryptanalysis [2] indicated that the BN_P256 curve is not as secure as previously

thought, but this was still used in our tests as the other curve (BN_P638) is not implemented in the Infineon TPM, or in the TPM emulator. Using the BN_P256 EC curve fixes the security parameter, $t(= 256)$, the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ and as a consequence the values of p and n as well.

The choice of generators for $\mathbb{G}_1$ and $\mathbb{G}_2$ is constrained. P_1, the generator for $\mathbb{G}_1$ is fixed in the TPM specification, while that for P_2 is fixed by AMCL.

The hash function, H_k is fixed as SHA256 and other hash functions are obtained by calculating SHA256 for the data and then taking the result $\pmod p$ for H_p and $\pmod n$ for H_n.

5 THREAT MODEL

5.1 Component Model

Our ECC-DAA scheme consists of four physical components, a HOST and its TPM, an ISSUER and a VERIFIER. Recall that a PLATFORM is the combination of a HOST with a TPM. The TPM is responsible for securely deriving keys and providing other cryptographic functions for the HOST. While a TPM is assumed to be tamper-resistant, we nevertheless allow some private keys which are held by TPMS or the ISSUERS to be potentially revealed as part of an attack.

5.2 Channel Model

Our ECC-DAA scheme operates over two channels:

(1) TPM ↔ HOST
(2) HOST ↔ ISSUER and VERIFIER

While a TPM is a HOST's RoT and typically embedded into its motherboard, the TPM remains a separate entity. Communication between the HOST and the TPM is over the Low Pin Count bus and following Camenisch *et al.* [12], we assume that there is a perfectly secure channel between the HOST and the TPM, *i.e.*, an adversary has no control over the channel and does not know it is being used. In the same paper, no constraints are imposed on the channel between the HOST and the ISSUER which means that it is under the control of an adversary and we also make this assumption. We return to the subtleties regarding these precise assumptions and their formal modelling in Section 6.3.

The channel model assumes a classical Dolev-Yao (DY) [22] adversary that can intercept, block, replay, spoof and send messages on the channel from any source. The DY adversary also learns the content of all messages unless they are cryptographically protected in which case the DY can only decrypt them if the decryption keys are known or can be derived.

6 TAMARIN SECURITY PROOFS

6.1 Methodology

The TAMARIN PROVER [35] is a state-of-the-art protocol verification tool for symbolic modelling. It supports unbounded verification, mutable global state, and flexible user-defined equational theories. Protocols are modelled using multi-set rewriting rules and properties are specified using first-order logic. The tool offers automatic verification succeeding in many cases, as well as an interactive verification mode for manual proof tree traversal. The tool provides both proofs, and disproofs by counter-example. Moreover, the TAMARIN PROVER models all communication using the standard DY adversary model.

We faithfully represent the protocol specification given in Section 3 in TAMARIN models. The naming convention used in these models follows closely the names of the TPM commands as well as the variable names and labels used in Figure 1 through Figure 7. Our models are based on the approach of Whitefield *et al.* [46] for their modelling of the DAA scheme specified by ISO/IEC-20008-2:2013 mechanism 4 [29].

Due to the fine-grained modelling of our ECC-DAA scheme, the computational complexity of verifying it using the TAMARIN PROVER proved challenging even on a server with 2 Intel Xeon E5-2667 CPUs (16 cores, 32 threads) and 378GB of RAM. In order to reduce the computational complexity of our models, we separate the analysis into three parts: a) JOIN, SIGN and VERIFY; b) JOIN, CERTIFY and VERIFY; and c) JOIN, QUOTE and VERIFY. Each of these will have two models: with or without a basename set. All of them share copies of the same JOIN operation as the TAMARIN PROVER does not support including shared code.

For example, the TAMARIN models for our ECC-DAA scheme that model the JOIN, SIGN and VERIFY operations, require almost 1400 lines of code consisting of over 38 rules. To model the interactions between a TPM, a HOST, an ISSUER and a VERIFIER requires almost 1000 lines of code. The remainder is used to specify lemmas which encapsulate the security properties our model satisfies. To our knowledge, this is the first time that all the security properties of an ECC-DAA scheme have been expressed formally.

Note that we do not discuss the SETUP and LINK operations. The SETUP operation is an off-line process and does not need to be modelled. The LINK operation simply compares certain values in two signatures and returns true or false depending on whether these values match or not. This functional property is captured within our TAMARIN model as part of functional correctness but not discussed in more detail in the paper.

The TAMARIN PROVER is then used to verify our scheme's security properties (*SP1* to *SP7*, as described in Section 3.2) for these models. *SP1*, *SP2*, *SP3*, *SP4* and *SP7* are encoded using first-order logic formulae and prefixed by the lemma identifier in the models. The formulae are then evaluated by the TAMARIN PROVER over runs of the scheme defined in the models. In order to verify the anonymity (*SP5*) and user-controlled unlinkability (*SP6*) properties, we require a further twelve variants since their proofs use the TAMARIN PROVER's Observational Equivalence proof mode [4], instead. *SP5* and *SP6* are considered to be indistinguishability properties and Observational Equivalence is the standard way of proving those in a symbolic model. In this mode, the TAMARIN PROVER reasons about two systems (for example, two instances of a protocol), by showing that a DY adversary cannot distinguish these two systems. While it would have been desirable to prove these two properties without these additional models, due to the high number of rules in our first-order logic models, the resultant state space is too large for exploration in the TAMARIN PROVER's observational equivalence proof mode. Therefore, to prove these properties, the 12 variants for this mode use fewer rules. This is achieved by collapsing the respective JOIN & SIGN, JOIN & CERTIFY, and JOIN & QUOTE operations and treating the HOST and its TPM as a single entity for these 12 variants only. This simplification only removes the model's fine-grained mapping

Table 1: 18 Tamarin model variants

	Tamarin Prover's Proof mode					
	First-order Logic		Obs. Equiv.			
Context	$bsn \neq \perp$	$bsn = \perp$	$bsn \neq \perp$		$bsn = \perp$	
Sign	*SP1*, *SP2*, *SP3* *SP4*, *SP7*	*SP1*, *SP3*, *SP7*	*SP5*	*SP6*	*SP5*	*SP6*
Certify	*SP1*, *SP2*, *SP3* *SP4*, *SP7*	*SP1*, *SP3*, *SP7*	*SP5*	*SP6*	*SP5*	*SP6*
Quote	*SP1*, *SP2*, *SP3* *SP4*, *SP7*	*SP1*, *SP3* *SP7*	*SP5*	*SP6*	*SP5*	*SP6*

to the Tpm calls but does not restrict the knowledge a DY adversary gains by observing the protocol. Table 1 shows an overview of all the different models and which security properties are included in each.

Due to the complexity of encoding all the calls between a Host, its Tpm, the Issuer and a Verifier in our models, the standard heuristic employed by the Tamarin Prover to prove the model's properties only works for one of our security properties (*SP2*). All other security properties required the assistance of a *proof oracle* that reorders the ranking of the open proof goals in the Tamarin Prover, and thus enabling it to pick the ones which will result in the desired proof. The oracles only influence termination, not the result as such, so soundness is preserved. These oracles were constructed by using the web interface of the Tamarin Prover and stepping through these proofs. The steps were then encoded into a Python script of the order of about 100 lines of code (see Chapter 9 of the Tamarin manual [41]). In our models, the lemmas that require the use of such an oracle are labelled with the prefix "oracle_" while the one without the need for an oracle has the prefix "auto_".

Our 18 Tamarin models and associated oracles are available at [19] together with instructions on how to reproduce the results.

6.2 Symbolic Modelling

The Tamarin Prover provides symbolic analysis, *i.e.*, instead of using finite field elements or points of an elliptic curve we consider abstract terms. For example, symmetric encryption of a message m under the key k is given by $senc(m, k)$. When processing a model, the Tamarin Prover operates under the *perfect cryptography assumption*. In other words, the DY adversary can only decrypt messages for which she knows the key, she cannot brute-force a key or mathematically derive it, nor can she "undo" a hash of a message or find a hash collision. However, she can manipulate terms symbolically, *i.e.*, build up new terms by combining existing ones or computing new ones using her knowledge of the protocol.

Our Ecc-Daa scheme's security relies on verifying numerous mathematical equalities involving complex terms using a combination of bilinear maps, finite field arithmetic and hashes. In order to ensure the validity of the computations, we introduced our own equational theories whose purpose is to verify the correct construction of terms in order to simplify them according to their mathematical properties.

As we have stated above there are 18 Tamarin models and they are partitioned into two classes: those representing schemes that have a basename set ($bsn \neq \perp$), and those which do not set a basename ($bsn = \perp$). This means that during the analysis an adversary is not able to produce signatures without basenames in models with basenames and vice versa. This arguably limits the capabilities of the adversary and the results of the symbolic modelling are thus not in the context of the most general adversary.

6.3 Ecc-Daa Modelling Choices

Separation of Host and Tpm: our Ecc-Daa scheme relies on a Host and its Tpm which are co-located but distinct components. As such both the Host and Tpm are modelled as separate entities, and the binding of a Tpm to a Host constitutes a Platform. In our models there is a one-to-one correspondence between a Host and a Tpm, *i.e.*, a Host has only one Tpm and that Tpm is bound to that Host only. Modelling this separation allows for the Host and its Tpm to be treated independently, in particular it allows for the model to explore scenarios where the Host or Tpm or both are compromised.

Secure Channel between the Host *and its* Tpm: following the channel model described in Section 5.2, the communication between the Host and the Tpm has been modelled as a secure channel, *i.e.*, a channel which the DY adversary cannot observe nor interfere with. While this is a very strong restriction on the capabilities of the DY adversary, it reflects the current security assumption around the interaction between the Host and its Tpm. Moreover, having modelled this channel separately, it also allows for some of these assumptions to be relaxed individually in future.

Issuer *and* Verifier: both these parties have been modelled as independent entities in the models. It is important to note that the models allow the Issuer to be compromised, *i.e.*, its private keys can be leaked, while there is no such requirement on the Verifier as it does not handle any sensitive information. Both Issuer and Verifier communicate with the Host over an insecure channel which is fully controlled by the DY adversary. Note that the communication between the Host and the Issuer during the Join phase is usually assumed to use a secure and authenticated channel [29]. However, establishing such a channel when the Host and Issuer are not co-located is difficult and if the Join phase of our scheme can be verified to work using an insecure channel, then it will also work when a secure and authenticated channel can be established. Modelling an insecure channel between a Host and a Verifier is, however, a realistic assumption since a Verifier can be any third party who has obtained, through any channel, a given signed message, certified key or quoted PCR values and wants to establish the data's integrity and authenticity.

6.4 Formalisation of Security Properties

Having modelled our Ecc-Daa scheme using the Tamarin Prover's multi-set rewriting rules, we now formalise its security properties. Notably, *SP7* highlighted a weakness in the Join phase of our Ecc-Daa scheme which was easily rectified. This fix, however, is critical since a successful Join phase is required by all the other security properties since they are concerned with the security or indistinguishability of the messages created by a Platform after it has successfully completed the Join phase of our scheme.

Some notes on the notation used in Tamarin's lemmas: "All" is Tamarin's version of the universal quantifier, ∀. The logical operators AND (∧), NOT (¬) and IMPLIES (⟹) are expressed using "&", "*not*()" and "==>" respectively. Temporal variables are prefixed with a "#", *e.g.*, "#*t*01", and "@" indicates a specific time point, *e.g.*, "@ *t*01". Anything following "//" or in between "/* */" is a comment and therefore ignored by Tamarin.

SP1: Correctness: This property requires the Tamarin Prover to find that there exists an execution (called a trace in Tamarin) of our scheme in which no private keys have been revealed and all the steps in the protocol are executed in the correct order and with the right parties. Consequently, the Tamarin Prover needs to prove that our Ecc-Daa scheme allows a given Platform to go through its Join phase with an Issuer successfully and using the credentials it obtained during that phase to then execute the Sign, Certify and Quote operations to generate a message, *e.g.*, signature, certified key, or some quoted PCR values, which a Verifier can check using the corresponding Verify operation. In our Tamarin models this property is captured by all lemmas whose names are prefixed with "oracle_correctness" and they all proved successfully.

SP2: User-controlled linkability: This property states that for a given single basename *bsn*(≠ ⊥), a DY adversary finds it hard to create two different messages, *sigma*1 and *sigma*2, under the same private key, *e.g.*, $f1$, and basename, *bsn*, but the messages are not linked. Since the "hardness" of a problem is related to its computational complexity, we need to recast this statement as an assertion on the messages and their private keys. Consequently, we encode this property in Tamarin by stating that given two messages, which are not linked but which used the same basename, can only have been created if the private keys used in their creation were different to start with. The lemma shown in Listing 1 shows our implementation.

```
lemma auto_SP2_UserControlledLinkability:
" All bsn sigma1 sigma2 f1 f2 #t01 #t02 #t03.
//given 2 signatures that are not linked
UnlinkedSignatures(sigma1, sigma2) @ t01
& not(sigma1=sigma2) // and different
//but used the same base name and
//a private f1 and f2, respectively
& VerifiedSignatureDeAnonymised(bsn, sigma1, f1) @ t02
& VerifiedSignatureDeAnonymised(bsn, sigma2, f2) @ t03
==> //then f1 is not the same as f2
not(f1=f2) "
```

Listing 1: Linkability

Note that this lemma is only applicable to the variants of our models in which a basename can be set. In order to prove *SP2*, we defined an action label, `VerifiedSignatureDeAnonymised`, inside the rule called `Verifier_Check_TPM_Message` responsible for checking the validity of a message. This action label recovers the private key associated with that message and thus allows us to reason about the private keys used in the message. This should obviously not be possible according to the mathematical properties of the Ecc-Daa scheme, however, we only used this to demonstrate that any two valid signatures which are not linked must have used distinct private Ecc-Daa keys, $f1$ and $f2$. This lemma proves in all applicable variants of our model.

SP3: Unforgeability: This property states that a DY adversary, who is in the possession of a set of Platforms' private keys and associated credentials, finds it hard to forge a valid message for a private key and credential, which is not in that set. In order to encode this property into a lemma for the Tamarin Prover, we again need to recast it to be suitable for symbolic analysis. As a result, our lemma states that if there exists a valid message m associated with a Platform P, but P did not send it, then P's credentials must have been compromised and the adversary used them to create m. This property holds in the two variants of our models, *i.e.*, when a basename can be set, $bsn \neq \perp$, and when the basename is unset, $bsn = \perp$.

Our proof shows that a verifiable message either originated from a Platform P, which has not been compromised, *i.e.*, whose keys have not leaked, or from a DY adversary in possession of P's private keys. In other words, a DY adversary cannot construct a valid message for a Platform P any other way than by knowing P's private keys.

SP4: Non-frameability: This property states that no combination of dishonest Issuers and Platforms can create a valid message, m_0, which can be linked to some given message, m_1, generated by an honest Platform, unless that Platform produced the message, m_0. In our model this needs to be recast into a positive assertion which we capture in a lemma that states when given a Platform P whose private keys have not been revealed and which sent a valid message m_1 that can be linked to a second valid message m_0 then P must have also sent the second message m_0. This property does not apply to the models in which the basename is not set since linking of messages requires a basename to be set.

SP5: Anonymity: Anonymity in our scheme is the requirement that a DY adversary, who does not know a Platform's private key, finds it hard to recover the identity of the Tpm used by the Platform from a given message. In the Tamarin Prover this property can be expressed using *Observational Equivalence* properties (*cf.* Section 6.1).

In our models, we provide the adversary with the public keys of two distinct Tpms as well as two messages, *sigma*1 and *sigma*2 where one Tpm created *sigma*1 while the other Tpm created *sigma*2. Using its *Observational Equivalence* mode, the two systems that the Tamarin Prover needs to distinguish differ by the message the adversary is shown. In the first system, the adversary gets presented with, for example, *sigma*1, while in the second system *sigma*2 is used, instead. If the adversary can associate at least one of the public Tpm keys with a given message but not the other message, then the two systems differ and the anonymity will be broken. Therefore, if the Tamarin Prover cannot distinguish these two systems then the adversary is unable to tell the messages apart and the anonymity of the signatures are preserved. This lemma proves in all 6 variants of our model with and without a basename.

SP6: User-controlled unlinkability: User-Controlled unlinkability is the requirement that the user's identity is not revealed through her messages. In the Tamarin Prover this is again expressed as an *Observational Equivalence*. For this property, our Tamarin model generates either two messages with random basenames from one Platform, or two messages with random basenames from two different Platforms. The Tamarin Prover then checks if those two systems can be distinguished. In other words if it were possible to spot that the two signatures come from the same Tpm or that they came from different Tpms then that would break the desired

unlinkability. This lemma proves in all 6 variants of our model with and without a basename.

SP7: Authentication - Injective Agreement during Join*:* We describe the authentication security property which initially did not hold and required an amendment to our Ecc-Daa scheme. Whitefield *et al.* [46] in their analysis of the ISO/IEC-20008-2:2013 mechanism 4 Ecc-Daa scheme [29] demonstrated an attack which showed that the standard did not satisfy the weak agreement property of Lowe's hierarchy of authentication specifications [33]. This hierarchy is the de-facto standard used for authentication properties in symbolic verification. Their proposed fix for the attack is to include the public endorsement key of the Tpm, $\mathcal{E}$, in the proof of knowledge, σ_{ch} (*cf.* Figure 1). We incorporated their fix into our scheme because without it our scheme suffers from the same attack. Indeed, with the fix our Ecc-Daa scheme satisfies Lowe's stronger injective agreement property which in our setting translates to: whenever a Platform P, identified by its Tpm's public endorsement key, $\mathcal{E}$, completes a run of the Join operation, apparently with the Issuer I, then I has previously been running the protocol, apparently with Platform P, and I was acting as an Issuer in its run, and both P and I agree on the exchanged credentials. Note that this lemma proves in the six variants of our model even though we assume that the Join operation between a Platform and an Issuer is executed over an insecure channel. This gives a stronger result than assuming a secure, authenticated channel as per the ISO standard and demonstrates the power of the symbolic verification.

To summarise, the formalisation and proofs of the security properties required utilising two distinct proof methodologies, *i.e.*, Observational Equivalence proofs versus first order logic statements, which required the creation of separate models for each one. Moreover, the complexity of some of the proofs meant that additional models were needed all of which impacts on the capability of the adversary to observe and create messages. However, given these constraints, we have shown that all of our security properties can be proven given a suitable model of our Ecc-Daa scheme.

7 CONCLUSION AND FUTURE WORKS

In this paper we have presented the fine-grained details of Tpm 2.0 API calls needed to implement Chen *et al.* 's scheme [16] and extended it to include Certify and Quote operations.

This research has developed numerous formal models to capture the details of this Ecc-Daa scheme at the level of Tpm 2.0 calls. Moreover, all Ecc-Daa security and authentication properties have been formalised and verified in a symbolic model. The verification of our models shows that Whitefield *et al.* 's fix [46] for their authentication attack is also required in Chen *et al.* 's scheme [16] and hence we recommend that the TCG incorporate this fix into a future errata of the Tpm specification.

To reduce the complexity of the formal analysis, our proofs were done in variants of our model. However, the use of variants of a symbolic model can reduce the adversarial capabilities. Our future research will therefore look at approaches to decompose our model in a way which preserve the adversarial capabilities while keeping the complexity of the analysis within the tool's capacity.

Note that these properties were proved in the context of where the adversary could not control the channel between a Host and its Tpm as their physical co-location should make the channel between them secure. While the Tpm itself is protected, the channel provides no protection and is accessible with additional equipment. Therefore, in future, we will consider a stronger adversary with physical access to Platforms on which it can perform side-channel attacks without compromising the Tpm, *i.e.*, extract any private keys generated and stored within a Tpm. This might include modelling a passive adversary who can only eavesdrop on the data items transferred between the Host and its Tpm and an active adversary who can additionally inject, spoof and block data items on that channel.

In addition to symbolic analysis tools such as Tamarin Prover, verifiers such as CryptoVerif [5] or EasyCrypt [3] have also been successful in formally analysing computational proofs and complementing our research with mechanised computational proofs would also be another interesting area of future research.

We implemented the Ecc-Daa scheme in C++ using the Tpm 2.0 specification and a hardware Tpm. To the best of our knowledge, this is the first open-source implementation of an Ecc-Daa scheme using a Tpm. Our implementation has shown significant differences in the signing performance when the basename is empty or set, and understanding this metric is important when designing engineering solutions. In particular, requiring signatures to be linked is an expensive operation. Yang *et al.* [47] have recently proposed an optimised Daa scheme so that the signing performance is more comparable when the basename exists and when it is empty. One interesting future work would be to explore a formal analysis of their optimised Daa scheme.

Providing a symbolic model as well as a reference implementation for Ecc-Daa is very timely, since recent novel application areas for Daa have been identified, such as web authentication systems, *e.g.*, FIDO [32], and Intelligent Transportation Systems (ITS) [20, 45]. Our reference implementation has been used in a Personal Activity and Health Data Tracking use case as produced within the context of the FutureTPM project [24]. Another avenue for future research would be to apply our Ecc-Daa implementation to a Vehicle-to-Anything (V2X) communication use case for ITS. V2X is required to be privacy-preserving [45], and applying our Ecc-Daa scheme would enable ITS entities to generate and certify their own anonymous credentials. These credentials can be used for the signing of V2X messages, and other entities can verify the messages without prior knowledge. One further advantage of our Ecc-Daa scheme would also address establishing the trustworthiness of ITS entities using "*remote attestation*" [43]. For example, in ITS vehicles will be largely controlled and operated by a wide array of software, and being able to attest, the software components executing on vehicles is of critical importance. Measurements of the software a vehicle is running can be stored in the Tpm's PCR, and an Ecc-Daa signature produced.

Acknowledgements: The research was partly funded by the following EPSRC projects: Improving customer experience while ensuring data privacy for intelligent mobility - EP/N028295/1, Privacy Enhanced Capabilities for VANETs using Direct Anonymous Attestation Project - EP/R511791/1, and the European Union's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM).

REFERENCES

[1] Tolga Acar, Lan Nguyen, and Greg Zaverucha. 2013. A TPM Diffie-Hellman Oracle. *IACR Cryptology ePrint Archive* 2013 (2013), 667.

[2] Razvan Barbulescu and Sylvain Duquesne. 2018. Updating Key Size Estimations for Pairings. *Journal of Cryptology* (29 Jan 2018). https://doi.org/10.1007/s00145-018-9280-5

[3] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. 2013. EasyCrypt: A Tutorial. In *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*. 146–166. https://doi.org/10.1007/978-3-319-10082-1_6

[4] David Basin, Jannik Dreier, and Ralf Sasse. 2015. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1144–1155.

[5] Bruno Blanchet. 2007. Cryptoverif: Computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied*, Vol. 117.

[6] Ernie Brickell, Jan Camenisch, and Liqun Chen. 2004. Direct Anonymous Attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*. ACM, New York, NY, USA, 132–145. https://doi.org/10.1145/1030083.1030103

[7] Ernie Brickell, Liqun Chen, and Jiangtao Li. 2008. A New Direct Anonymous Attestation Scheme from Bilinear Maps. In *Trusted Computing - Challenges and Applications, First International Conference on Trusted Computing and Trust in Information Technologies, Trust 2008, Villach, Austria, March 11-12, 2008, Proceedings*. 166–178. https://doi.org/10.1007/978-3-540-68979-9_13

[8] Ernie Brickell, Liqun Chen, and Jiangtao Li. 2009. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. *International journal of information security* 8, 5 (2009), 315–330.

[9] Ernie Brickell, Liqun Chen, and Jiangtao Li. 2011. A (Corrected) DAA Scheme Using Batch Proof and Verification. In *Trusted Systems - Third International Conference, INTRUST 2011, Beijing, China, November 27-29, 2011, Revised Selected Papers*. 304–337. https://doi.org/10.1007/978-3-642-32298-3_20

[10] Ernie Brickell and Jiangtao Li. 2012. Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. *IEEE Trans. Dependable Sec. Comput.* 9, 3 (2012), 345–360. https://doi.org/10.1109/TDSC.2011.63

[11] Daniel R. L. Brown and Robert P. Gallant. 2004. The Static Diffie-Hellman Problem. *IACR Cryptology ePrint Archive* 2004 (2004), 306.

[12] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. 2017. One TPM to Bind Them All: Fixing TPM 2.0 for Provably Secure Anonymous Attestation. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 901–920.

[13] Jan Camenisch, Manu Drijvers, and Anja Lehmann. 2016. Universally Composable Direct Anonymous Attestation. In *Public Key Cryptography (2) (Lecture Notes in Computer Science)*, Vol. 9615. Springer, 234–264.

[14] Liqun Chen. 2009. A DAA Scheme Requiring Less TPM Resources. In *Information Security and Cryptology - 5th International Conference, Inscrypt 2009, Beijing, China, December 12-15, 2009. Revised Selected Papers*. 350–365. https://doi.org/10.1007/978-3-642-16342-5_26

[15] Lily Chen. 2009. *Recommendation for Key Derivation Using Pseudorandom Functions*. SP 800-108. National Institute for Standards and Technology.

[16] Liqun Chen and Jiangtao Li. 2013. Flexible and scalable digital signatures in TPM 2.0. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. 37–48. https://doi.org/10.1145/2508859.2516729

[17] Liqun Chen, Paul Morrissey, and Nigel P. Smart. 2009. DAA: Fixing the pairing based protocols. Cryptology ePrint Archive, Report 2009/198. https://eprint.iacr.org/2009/198.

[18] Liqun Chen, Chris Newton, Ralf Sasse, Helen Treharne, Stephan Wesemyer, and Jorden Whitefield. 2020. Ecc-Daa C++Implementation. GitHub. https://github.com/UoS-SCCS/ecc-daa.

[19] Liqun Chen, Chris Newton, Ralf Sasse, Helen Treharne, Stephan Wesemyer, and Jorden Whitefield. 2020. Ecc-Daa Tamarin models. GitHub. https://github.com/tamarin-prover/tamarin-prover/tree/develop/examples/asiaccs20-eccDAA.

[20] Liqun Chen, Siaw-Lynn Ng, and Guilin Wang. 2011. Threshold Anonymous Announcement in VANETs. *IEEE Journal on Selected Areas in Communications* 29, 3 (2011), 605–615. https://doi.org/10.1109/JSAC.2011.110310

[21] Liqun Chen, Dan Page, and Nigel P. Smart. 2010. On the Design and Implementation of an Efficient DAA Scheme. In *CARDIS (Lecture Notes in Computer Science)*, Vol. 6035. Springer, 223–237.

[22] Danny Dolev and Andrew Chi-Chih Yao. 1983. On the security of public key protocols. *IEEE Trans. Information Theory* (1983).

[23] Free Software Foundation, Inc. 2019. GCC, the GNU Compiler Collection. https://gcc.gnu.org [Online; accessed 12-June-2019].

[24] FutureTPM. 2018-2020. Future Proofing the Connected World: A Quantum-Resistant Trusted Platform Module (EU H2020 Project, Grant Agreement No. 779391).

[25] Infineon Technologies AG. 2017. Iridium SLB 9670 TPM2.0 Linux. https://www.infineon.com/cms/en/product/evaluation-boards/iridium9670-tpm2.0-linux/ [Online, accessed 03-May-2019].

[26] International Business Machines. 2017. IBM's Software TPM 2.0 Version 1119. https://sourceforge.net/projects/ibmswtpm2 [Online, accessed 03-May-2019].

[27] International Business Machines. 2017. IBM's TPM 2.0 TSS Version 1119. https://sourceforge.net/projects/ibmtpm20tss/ [Online, accessed 03-May-2019].

[28] ISO/IEC 11770-4:2017 2017. *Information technology – Security techniques – Key management – Part 4: Mechanisms based on weak secrets*. Standard. International Organization for Standardization, Geneva, CH.

[29] ISO/IEC 20008-2:2013 2013. *Information technology - Security techniques - Anonymous digital Signatures Part 2: Mechanisms using a group public key*. Standard. International Organization for Standardization, Geneva, CH.

[30] Jakob Jonsson and Burt Kaliski. 2003. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. RFC 3447. RFC Editor. http://www.rfc-editor.org/rfc/rfc3447.txt

[31] Vireshwar Kumar, He Li, Noah Luther, Pranav Asokan, Jung-Min (Jerry) Park, Kaigui Bian, Martin B. H. Weiss, and Taieb Znati. 2018. Direct Anonymous Attestation with Efficient Verifier-Local Revocation for Subscription System. Cryptology ePrint Archive, Report 2018/290. https://eprint.iacr.org/2018/290.

[32] Rolf Lindemann, Jan Camenisch, Manu Drijvers, Alec Edgington, Anja Lehmann, and Rainer Urian. 2017. FIDO ECDAA Algorithm. https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html

[33] Gavin Lowe. 1997. A Hierarchy of Authentication Specification. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*. 31–44.

[34] Noah Robert Luther. 2017. *Implementing Direct Anonymous Attestation on TPM 2.0*. Master's thesis. Virginia Tech. https://vtechworks.lib.vt.edu/handle/10919/86349

[35] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. 696–701. https://doi.org/10.1007/978-3-642-39799-8_48

[36] OpenSSL. 2017. OpenSSL Cryptography and SSL/TLS toolkit. https://www.openssl.org/ [Online, accessed 03-May-2019].

[37] TCG. 2014. *EK Credential Profile: For TPM Family 2.0*. Rev 2.0. Trusted Computing Group.

[38] TCG. 2016. *Trusted Platform Module 2.0 Library Specification*. Rev. Trusted Computing Group. https://trustedcomputinggroup.org/resource/tpm-library-specification/

[39] TCG. 2016. *Trusted Platform Module 2.0, Part 1: Architecture*. Rev 1.38. Trusted Computing Group.

[40] TCG. 2016. *Trusted Platform Module 2.0, Part 2: Structures*. Rev 1.38. Trusted Computing Group.

[41] The Tamarin Team. 2016. Tamarin prover manual. https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf [Online: accessed 09-April-2019].

[42] The Apache Software Foundation. 2019. The Apache Milagro Cryptographic Library. https://github.com/apache/incubator-milagro-crypto/ [Online, accessed 03-May-2019].

[43] The Trusted Computing Group. 2016. TPM 2.0 Library Specification. https://trustedcomputinggroup.org/resource/tpm-library-specification/ [Online, accessed 11-April-2019].

[44] Trusted Computing Group. 2018. Trusted Computing. https://trustedcomputinggroup.org/trusted-computing/ [Online; accessed 27-June-2018].

[45] Jorden Whitefield, Liqun Chen, Thanassis Giannetsos, Steve Schneider, and Helen Treharne. 2017. Privacy-enhanced capabilities for VANETs using direct anonymous attestation. In *2017 IEEE Vehicular Networking Conference, VNC 2017, Torino, Italy, November 27-29, 2017*. 123–130. https://doi.org/10.1109/VNC.2017.8275615

[46] Jorden Whitefield, Liqun Chen, Ralf Sasse, Steve Schneider, Helen Treharne, and Stephan Wesemyer. 2019. A Symbolic Analysis of ECC-based Direct Anonymous Attestation. In *IEEE 4th European Symposium on Security and Privacy*.

[47] Kang Yang, Liqun Chen, Zhenfeng Zhang, Chris Newton, Bo Yang, and Li Xi. 2018. Direct Anonymous Attestation with Optimal TPM Signing Efficiency. Cryptology ePrint Archive, Report 2018/1128. https://eprint.iacr.org/2018/1128.

A APPENDIX

A.1 The Activate Credential Procedure

At the start of the procedure to issue a credential, the Issuer receives:

(1) $\mathcal{E}$ – information about the public RSA endorsement key.
(2) Q_{PD} – the public data for the Ecc-Daa key.

The Issuer generates an attestation key credential, C (to use with the Ecc-Daa key) together with a random credential key, $\mathcal{K}$. This key, $\mathcal{K}$, is used to encrypt the attestation key credential, C and the

activate credential process used to make sure that $\mathcal{K}$ can only be used by the Host if:

(1) $\mathcal{E}$ comes from a Tpm used by the Host.
(2) The Daa key was generated by the same Tpm.

To do this the Issuer

(1) Validates $\mathcal{E}$. In the simplest case this can be done by checking $\mathcal{E}$ against a list of valid public endorsement keys. More generally this will be done by providing a certificate for the key that is validated by the Tpm's manufacturer. If $\mathcal{E}$ is valid the Issuer can assume that $\mathcal{E}$ has the correct properties to 'unwrap' the credential blob (described below) that will be generated. The endorsement key should be an RSA storage key and to align with the TCG endorsement key profile [37] this key will be a 2048 bit RSA key, that will use AES 128 as its symmetric encryption algorithm and SHA256 as its hash algorithm.
(2) Confirms that the Ecc-Daa key has the necessary properties for the credential being issued. The Ecc-Daa key should be an restricted ECC signing key that is fixed to the Tpm and fixed to its parent. Object properties are detailed in Part 1 of the Tpm 2.0 specifications (pg. 166) [39]. In our case the key will use SHA256 as its underlying hash algorithm.
(3) Calculates the name of the Ecc-Daa key. This name is derived from the key's public data, for details see Part 1 (pg. 124) and Part 2 (pg. 135) of the Tpm specifications [39, 40].

$$Q_{\text{N}} = \text{nameAlgID}_{16} \parallel H_{name}(Q_{\text{PD}})$$

nameAlgID$_{16}$ is the 16-bit identifier for the hash algorithm, H_{name} (0x000b for SHA256).
(4) Generates a random seed value, $s \leftarrow \{0,1\}^t$.
(5) Derives a symmetric encryption key, k_e, and an HMAC key, k_h using a key derivation function. These keys are used to protect the key $\mathcal{K}$. The seed and the name of the Ecc-Daa key are both parameters of the key derivation function (KDF). Using the name is important as it ties in the Ecc-Daa key. Details of the key derivation function are given in Part 1 of the Tpm 2.0 specifications (pg. 43) [39] and the NIST Special Publication 800-108 [15]. An additional parameter is used in the KDF to separate the different keys, for k_e it is "STORAGE", while for k_h it is "INTEGRITY" (both are null terminated strings and the null is included in the calculations). For our purposes we write:

$$\begin{aligned} k_e &= KDF(s, \text{“}STORAGE\text{”}, Q_{\text{N}}) \\ k_h &= KDF(s, \text{“}INTEGRITY\text{”}, NULL) \end{aligned}$$

(6) Encrypts $\mathcal{K}$ using the key, k_e, giving $\widehat{\mathcal{K}}$. This encryption is done using AES 128 in CFB mode with an IV of zero.

$$\widehat{\mathcal{K}} = \text{senc}(\mathcal{K}, k_e)$$

(7) Generates an HMAC, $\mathcal{H}$, for $\widehat{\mathcal{K}}$ using the key k_h. The HMAC also incorporates the key name, Q_{N}:

$$\mathcal{H} = \text{hmac}(k_h, \text{len}_{16}(\widehat{\mathcal{K}}) \parallel \widehat{\mathcal{K}} \parallel Q_{\text{N}})$$

(8) The encrypted key and the HMAC together form a 'credential blob', $\text{CB}=\mathcal{H} \parallel \text{len}_{16}(\widehat{\mathcal{K}}) \parallel \widehat{\mathcal{K}}$.
(9) Encrypts the seed s using $\mathcal{E}$, giving $\hat{s}$ – in the Tpm specification this is called the secret. The encryption uses RSA-OAEP encryption with SHA256 as the hash function and MGF1 padding. It also uses a null terminated label, "IDENTITY", see RFC3447 [30].

$$\hat{s} = \text{aenc}(s, \mathcal{E})$$

(10) Encrypts C using $\mathcal{K}$, giving $\widehat{C}$.

$$\widehat{C} = \text{senc}(C, \mathcal{K})$$

(11) Sends the encrypted attestation key credential, $\widehat{C}$, the credential blob, $\text{CB}=\mathcal{H} \parallel \text{len}_{16}(\widehat{\mathcal{K}}) \parallel \widehat{\mathcal{K}}$, and the secret, $\hat{s}$, to the Host.

In the Join operation diagrams (Figures 1 and 2) this process is abbreviated as:

$$\begin{aligned} [\,\text{CB}, \hat{s}\,] &= \text{make\_credential}(\mathcal{E}, Q_{\text{PD}}, \mathcal{K}) \quad \text{and} \\ \widehat{C} &= \text{senc}(C, \mathcal{K}) \end{aligned}$$

The Host uses the Tpm command to unwrap the credential blob and return the key, $\mathcal{K}$. To do this the Host loads the endorsement and Ecc-Daa keys into the Tpm and then uses TPM2_ActivateCredential to obtain the key, to do this the command:

(1) Decrypts the secret seed, s, using the private endorsement key.

$$s = \text{adec}(\hat{s}, e)$$

(2) Uses the seed together with Ecc-Daa key's name (which the Tpm already knows) to generate the encryption and HMAC keys that were used to create the credential blob.

$$\begin{aligned} k_e &= KDF(s, \text{“}STORAGE\text{”}, Q_{\text{N}}) \\ k_h &= KDF(s, \text{“}INTEGRITY\text{”}, NULL) \end{aligned}$$

(3) Uses these keys to check the integrity of the blob and decrypt the key, $\mathcal{K}$. To do this:
   (a) The credential blob is split into its component parts, $\mathcal{H}$ and $\text{len}_{16}(\widehat{\mathcal{K}}) \parallel \widehat{\mathcal{K}}$
   (b) The HMAC is calculated

$$\mathcal{H}' = \text{hmac}(k_h, \text{len}_{16}(\widehat{\mathcal{K}}) \parallel \widehat{\mathcal{K}} \parallel Q_{\text{N}})$$

   and checked against the value from the credential blob.
   (c) The credential key, $\mathcal{K}$, is calculated

$$\mathcal{K} = \text{sdec}(\widehat{\mathcal{K}}, k_e)$$

(4) The credential key is then returned to the Host.

On the protocol diagram this is written as:

$$\mathcal{K} = \text{activate\_credential}(\text{CB}, \hat{s})$$

The Host can then decrypt and use the attestation key credential, C.

$$C = \text{senc}(\widehat{C}, \mathcal{K})$$

In the Join operation this procedure is used twice. In the first case there is no attestation key credential to be encrypted, but the Issuer wants to check that the Host has access to the Tpm and can therefore unwrap the credential blob that it creates. In the second case the attestation key credential is the A, B, C and D values that form the Ecc-Daa key's attestation key credential together with the signature that the Host uses to validate the attestation key credential that it has received from the Issuer.

A.2 Attestation Data

Attestation data is generated by a number of Tpm commands. Here we are just considering the TPM2_Certify and TPM2_Quote commands. The attestation data is described in Part 1 of the Tpm 2.0 specifications (pg. 194) [39] with further details given in Part 2 of the Tpm 2.0 specifications (pg. 107 and pg. 109) [40]. The attestation data structures have the following fields:

(1) magic – a 32 bit number that is used to tag structures that are generated by a Tpm (referred to as the TPM_ GENERATED_VALUE, 0xFF544347).
(2) type – the type of the attestation structure.
(3) qualifiedSigner – the qualified name of the signing key. When using a Daa key this field is set to be an empty buffer.
(4) extraData – external information provided by the caller. This field should be set to the commitment data, c, but in current versions it is an empty buffer. This will be fixed in the next version of the Tpm standard.
(5) clockInfo – clock data.
(6) firmwareVersion – a 64 bit number identifying the firmware version.
(7) attested – data specific to the attestation type.
    For TPM2_Certify this is:
    (a) name – the name of the key being certified.
    (b) qualifiedName – the qualified name of the key being certified. When using a Daa key to certify a key this field is set to be an empty buffer.
    while for TPM2_Quote it is:
    (a) pcrSelect – information on the PCR values being selected and the hash algorithm being used.
    (b) pcrDigest – the digest of the selected PCR values.

Note that the clockInfo and firmwareVersion fields are not used directly in our Ecc-Daascheme.

A.3 Implementation Note

Writing protocols in terms of Tpm calls requires reading and understanding the Tpm 2.0 specification and this makes Tpm development challenging and causes a high-barrier of entry.

While the Tpm 2.0 specification [43] was designed to be easily maintainable, it is nevertheless challenging to read mainly due to its sheer size. It consists of over 1400 pages split into four parts which not only cover the core specifications, but also numerous errata covering the continuous development of the Tpm specification. Therefore, a particular Tpm will be based on the core specification and all of the relevant errata which it implements.

Once the relevant Tpm calls were identified it was found that the best way to proceed was to write test programs for these Tpm calls and to run them using the Tpm simulator. The Tpm simulator can always be reset and any mistakes rectified, this is not so easy with the actual device itself.

A.4 Timing Tests

The timing tests (sets of 50 runs) were repeated a number of times to see if there was a pattern to the occurrence of outliers. These tests were carried out both with, and without, resetting the Tpm, almost all data sets had outliers in varying positions. For example, Figure 9 shows the timing data for TPM2_Load for two different runs. Both runs show outliers, but in entirely different positions. The timings given in Tables 2 to 4 are for all of the runs that were undertaken with the outliers removed.

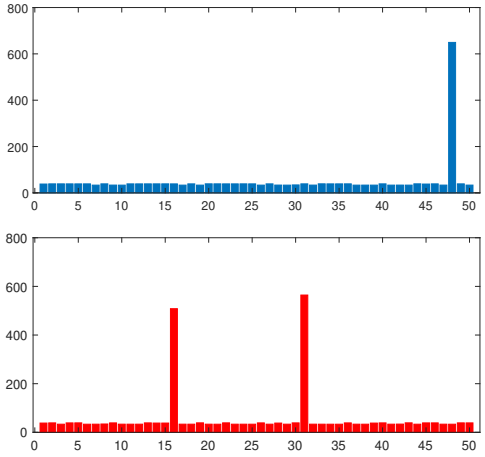

Figure 9: Two sets of 50 timings of TPM2_Load (times in ms).

Table 2: Timings for the Join operation

Time (ms)	M	SD
Host prepares	255.7	3.2
TPM2_Create	219.2	2.2
TPM2_Load	36.4	2.6
Issure challenges	2.6	0.3
Host responds	395.5	6.4
TPM2_ActivateCredential	219.4	3.2
TPM2_Commit	89.1	2.6
TPM2_Hash	25.1	2.6
TPM2_Sign	62.7	2.7
Issure verifies response	47.5	1.8
Issure creates AK credential	74.3	16.5
Host verifies AK credential	410.0	3.3
TPM2_ActivateCredential	219.4	3.2
Check pairings	148.4	0.7

A.5 Benchmarks

The protocols were implemented in a number of separate programs:

- The Setup and Join protocols: generate a random Ecc-Daa key and obtain its attestation key certificate;
- The Sign protocol: signing a message using the Ecc-Daa key and its randomised attestation key certificate;
- The Verify protocol: verify the signature and check the randomised attestation key certificate.

The Sign and Verify programs were run with and without a basename. If a basename was used it was a random string of 1 - 33 bytes.

Similar programs were written to test the Certify and Quote operations. The programs ran on a single thread, and opportunities for improving performance by multi-threading were ignored. The operations were implemented using readily available software packages rather than highly tuned code and, as such, they provide a baseline for future comparison. Prior to running the Certify operation, an

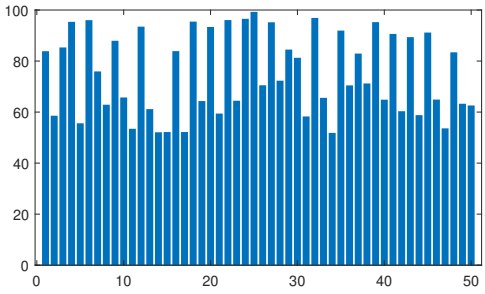

Figure 10: Timings for 50 runs of Issuer creates AK credential (times in ms).

Ecdsa key was generated and loaded into the Tpm. Similiarly, prior to running the Quote operation, we chose an arbitary but representative PCR to attest. Consequently, the Tpm was configured with PCR 23 (the application PCR) set to a known value. This is read and attested to by the Quote operation. Verification then checks the attestation data for the expected value and verifies the signature on the attestation data as well. In a real situation a number of PCR values in the Tpm will be set at boot time and Quote is then used to attest these values and confirm the state of the system.

Each operation was run 200 times and the times were measured using the C++ timing functions. The timings reported are for the operations running on a Raspberry Pi and Infineon Tpm. The inputs are random, and the times taken can therefore vary (see Figure 10 which shows the time taken for 'Issuer creates attestation key credential' procedure of the Join operation). The mean (M) and standard deviation (SD) of the results are reported. There were a few (< 6%) large outliers in some of the measurement sets (see Appendix A.4). However, as there seemed to be no pattern to their occurrence, these were removed before calculating the values that are shown in the tables. Some operations (for example, TPM2_ActivateCredential and 'Verify attestation key credential') occur in more than one of the protocols and the timings for these are averaged over all of the results rather than being reported separately.

Table 2 shows the timings for the Join operation. The times for the calculations carried out on the hardware Tpm are slow reflecting the use of a modest processor, *e.g.*, 219.4ms for TPM2_ActivateCredential. It is not clear what processor is used, or the Tpm's clock speed, and Infineon have not published this information. Note that on the Raspberry Pi processor significant time is taken for the pairings (*i.e.*, 148.4ms for Check pairings) – this is known to be a slow operation.

Table 3 shows the timings for the signing operations. It should be noted that with a basename the TPM2_Commit command takes much longer than without one (224.0ms vs 90.9ms), because with a basename the commit requires three exponentiations rather than just one (see Figure 3). Hence, using a basename, for example to link signatures, is an expensive operation. Note also that the times for the 'Host certifies' and 'Host quotes' operations are shorter than for the 'Host signs' operation as there is no need for the extra hash.

Table 3: Timings for the Sign operations

	$bsn = \perp$		$bsn \neq \perp$	
Time (ms)	M	SD	M	SD
Host commits	169.2	13.0	312.1	15.9
TPM2_Commit	90.9	1.1	224.0	1.0
Host signs	94.1	3.7	91.8	5.0
TPM2_Hash	26.7	1.5	26.2	2.0
TPM2_Sign	67.2	3.0	65.5	3.7
Host certifies	54.0	1.7	54.1	3.0
TPM2_Certify	53.9	1.7	53.9	3.0
Host quotes	53.3	2.3	53.8	2.0
TPM2_Quote	53.1	2.3	53.6	2.0

Table 4: Timings for Verify operation

	$bsn = \perp$		$bsn \neq \perp$	
Time (ms)	M	SD	M	SD
Verify AK credential	155.2	15.5	155.0	15.5
Verify signature	185.1	17.8	204.3	18.8
Verify Q_K certificate	184.4	16.4	203.2	19.0
Verify PCR quote	185.6	17.8	202.4	17.8

Table 4 presents the timings for the Verify operations. Note that verifying the attestation key credential (calculating the pairings) takes a significant amount of the verification time as this again uses the pairings which is a slow operation.

Timings for the laptop and Tpm emulator are not shown because all of the Tpm timings were around 88ms which may be due to obfuscation being used to mask Tpm commands in the simulator.