Direct Anonymous Attestation with Optimal TPM Signing Efficiency

Kang Yang[1,⋆], Liqun Chen[2], Zhenfeng Zhang[3,⋆⋆], Chris Newton[2], Bo Yang[3], and Li Xi[3]

[1] State Key Laboratory of Cryptology, Beijing 100878, China
yangk@sklc.org
[2] University of Surrey, UK
{liqun.chen, c.newton}@surrey.ac.uk
[3] Trusted Computing and Information Assurance Laboratory, SKLCS,
Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
{zfzhang, yangbo, xili}@tca.iscas.ac.cn

Abstract. Direct Anonymous Attestation (DAA) is an anonymous signature scheme, which is designed to allow the Trusted Platform Module (TPM), a small chip embedded in a host computer, to attest to the state of the host system, while preserving the privacy of the user. DAA provides two signature modes: fully anonymous signatures and pseudonymous signatures. To generate a DAA signature, the calculations are divided between the TPM and the host. One goal for designing new DAA schemes is to reduce the signing burden on the TPM as much as possible, since the TPM has only limited resources when compared to the host and the computational overhead of the TPM dominates the whole signing performance. In an optimal DAA scheme, the signing workload on the TPM will be no more than that required for a normal signature. DAA has developed about fifteen years, but no scheme has achieved this optimal signing efficiency for both signature modes. In this paper, we propose the first DAA scheme which achieves this optimal TPM signing efficiency for both signature modes. In particular, the TPM takes only a single exponentiation in a prime-order group when generating a DAA signature. Additionally, this single exponentiation can be precomputed, which enables our scheme to achieve fast online signing time.

Our DAA scheme is provably secure under the DDH, DBDH and q-SDH assumptions in the Universally Composable (UC) security model. Our scheme can be implemented using the existing TPM 2.0 commands, and thus is compatible with the TPM 2.0 specification. There are three important use cases for DAA: quoting platform configuration register values, certifying a key and signing a message. We have implemented and benchmarked the commands needed for these use cases on an Infineon TPM 2.0 chip. Based on these benchmark results, our scheme is about twice as fast as the existing DAA schemes supported by TPM 2.0 in terms of signing efficiency.

In addition, our DAA scheme supports selective attribute disclosure, which can satisfy more application requirements. We also extend our DAA scheme to support signature-based revocation and to guarantee privacy against subverted TPMs. The two extended DAA schemes keep the TPM signing efficiency optimal for both signature modes, and outperform existing related schemes in terms of signing performance.

Keywords: Cryptographic protocols · Direct anonymous attestation · TPM 2.0 implementation · Anonymous signatures · Provable security

1 Introduction

With the rapid growth of devices connected to the internet, it is becoming increasingly difficult to secure the devices [CCD+17]. To achieve better security, one approach is to place a root of trust such as a Trusted Platform Module (TPM) into such devices and use this to attest to the current state of the device. It is crucial that such attestations are privacy-preserving. On the one hand, anonymous attestation protects the privacy of owners of the devices, which adhere to one of the essential elements of privacy-enhancing systems

⋆ Supported by the National Natural Science Foundation of China (No. 61802021).

⋆⋆ Supported by the National Key Research and Development Program of China (No. 2017YFB0802504), and by the National Natural Science Foundation of China (No. U1536205).

(i.e., disassociability) developed recently by NIST [NIS15]. On the other hand, it minimizes the information available to the adversary.

The Trusted Computing Group (TCG), an industry standardization group, has developed Direct Anonymous Attestation (DAA) schemes to realize such attestations in a privacy-preserving manner. DAA schemes have been included in both of the TPM 1.2 and the TPM 2.0 specifications [Tru03,Tru16] and these have been adopted as ISO/IEC 11889 international standards [ISO09,ISO15]. The TPM 1.2 supports a RSA-based DAA scheme [BCC04] while the more recent TPM 2.0 supports multiple ECDAA schemes which are built on pairing-friendly elliptic curves. The ISO/IEC have standardized three DAA schemes [BCC04,CPS10,BL10b] in the ISO/IEC 20008-2 [Int13].

This paper will only focus on ECDAA schemes. Chen and Li [CL13] defined the TPM 2.0 commands needed to implement two alternative DAA schemes [CPS10,BL10b]. The flexibility of these commands means that they could be used to implement further ECDAA schemes. For example, the scheme presented by Camenisch et al. [CDL16a] (in the full version of their paper) can also be implemented using these TPM 2.0 commands. However, for this scheme the session identifiers used for the UC security proof need to be eliminated.

More than 500 million PCs shipped with TPM chips have been sold [Tru15], which makes DAA one of the most complex cryptographic schemes that has been widely implemented. In addition, for mobile devices, Raj et al. [RSW+16] presented the implementation of a firmware-based TPM (fTPM) using ARM TrustZone [ARM], which supports the TPM 2.0 specification. As a result, we can also implement the DAA schemes supported by TPM 2.0 in mobile devices.

DAA allows a TPM, which is a small chip embedded in a host, to attest either to the current state of the host system or to some other message, while preserving the privacy of the user owning the TPM. DAA provides two signature modes so that a user can decide whether a signature should be linkable to other signatures or not. Specifically, signatures w.r.t. the empty basename $\mathsf{bsn} = \perp$ are fully anonymous, i.e., unlinkable. Alternatively, signatures w.r.t. a non-empty basename $\mathsf{bsn} \neq \perp$ are pseudonymous, meaning that signatures under the same basename are linkable, while signatures under different basenames are unlikable. In some applications such as anonymous subscription [LDK+13,KLL+18] and vehicular communication (V2X) [PSFK15,WCG+17], pseudonymous signatures may be preferable or required for system operations. Pseudonymous signatures provide an advantage that allowing users to create pseudonyms at an internet service provider (ISP) and obtaining value-added services from the ISP.

The TPM is a small discrete chip and has only limited resources. In contrast, the host has much more powerful computational capability, e.g., the host is about a factor 300x faster than the TPM according to the experimental results [CL13,BCN14]. However, the host provides less security tolerance than the TPM. As pointed out by Camenisch et al. [CDL16b], the main challenge in designing a DAA scheme is to distribute the computational work between the TPM and host such that the workload of the TPM is as small as possible, while this does not affect the security in the case that the host is corrupted. A crucial feature of DAA is that the TPM and host cooperatively create a signature via executing a sign protocol. In an optimal DAA scheme, the signing workload on the TPM will be no more than that required for a normal signature such as ECSchnorr [Sch91,Tru16]. Specifically, only one exponentiation is required for the TPM when generating a signature, where one exponentiation is necessary to prevent the corrupted host from forging signatures without interacting with the TPM. Informally, we say that the TPM signing efficiency of a DAA scheme is *fully optimal* if the TPM costs only a single exponentiation per sign protocol execution for both two signature modes, and *partially optimal* if one exponentiation holds for only one signature mode.

The original DAA scheme was introduced by Brickell, Camenisch and Chen [BCC04], and has been standardized in TPM 1.2 [Tru03]. However, this scheme requires the TPM to compute exponentiations over a large RSA modulus, which leads to the costly computational burden for the TPM. Later, researchers resorted to bilinear pairings in order to construct more efficient ECDAA schemes. The ECDAA schemes fall into two categories: 1) LRSW-DAA schemes [BCL08,CMS08,CPS10,BFG+13,BCL12,CDL16b] based on the LRSW assumption [LRSW99] or its variants; and 2) SDH-DAA schemes [CF08,Che10,BL10b,CDL16a] based on the q-SDH assumption [BB08]. DAA schemes have developed about fifteen years, and improved gradually the signing efficiency of the TPM. However, only the LRSW-DAA schemes [BFG+13,BCL12] achieves the *partially optimal* TPM signing efficiency for fully anonymous signature mode. Furthermore, the best known SDH-DAA scheme [CDL16a] requires three exponentiations for the TPM to generate a signature for both two signature modes.

Chen and Urian [CU15] introduced DAA with attributes, which extends DAA to support attributes such as the manufacturer and model version of the platform and an expiration date of the credential etc. DAA with attributes supports selective attribute disclosure, i.e., a user can choose a part of attributes to disclose in a signature but other undisclosed attributes keep hidden. They proposed two DAA schemes with attributes by extending the LRSW-DAA scheme [CPS10] and the SDH-DAA scheme [BL10b] respectively, where their schemes allow the TPM to protect multiple attributes. While their SDH-DAA scheme with attributes [CU15] has $\mathcal{O}(1)$ credential size, their LRSW-DAA scheme with attributes [CU15] requires $\mathcal{O}(n)$ credential size, where n is the number of attributes. Later, Camenisch et al. [CDL16a] proposed a q-SDH-based DAA scheme with attributes, which stores all attributes on the host to obtain better efficiency. All these DAA schemes with attributes [CU15,CDL16a] can still be implemented using the TPM 2.0 commands.

Applications of DAA. We outline three types of DAA applications depending on what DAA signatures are used for.

Application I: a DAA signature is used to quote Platform Configuration Register (PCR) values. As the original application, DAA is used for remote attestation by quoting the PCR values recording the current state of the host system in order to protect the users' privacy. Besides, DAA with pseudonymous signature mode can be applied to V2X [WCG+17] via attesting to the current status of the vehicle such as its speed which is recorded in the PCR values.

Application II: a DAA signature is used to certify a key created by the TPM. One can apply DAA into the Fast IDentity Online (FIDO) authentication framework [FID17] to eliminate the unacceptably high risk in the FIDO basic attestation scheme that an attestation key is shared across a set of authenticators with identical characteristics. In this application, the TPM creates a new authentication key, and generates a fully anonymous signature (by cooperating with the host) to certify that the key is stored properly in the TPM. The FIDO alliance is in the process of standardizing a specification called FIDO ECDAA [CDE+17], which requires three exponentiations for the TPM to generate a signature.

Application III: a DAA signature is used to sign an arbitrary message given by the host such as a nonce from a verifier and a public key created by the host. In this case, DAA can be used to construct anonymous authentication by combining it with TLS [CLR+10], which prevents the sharing of credentials under the assumption that users cannot extract secret keys from TPMs. In addition, DAA is an appropriate cryptographic protocol for realizing anonymous subscription under the same assumption [KLL+18]. Public transportation system is also a potential application of DAA [ALT+15].

1.1 Our Contributions

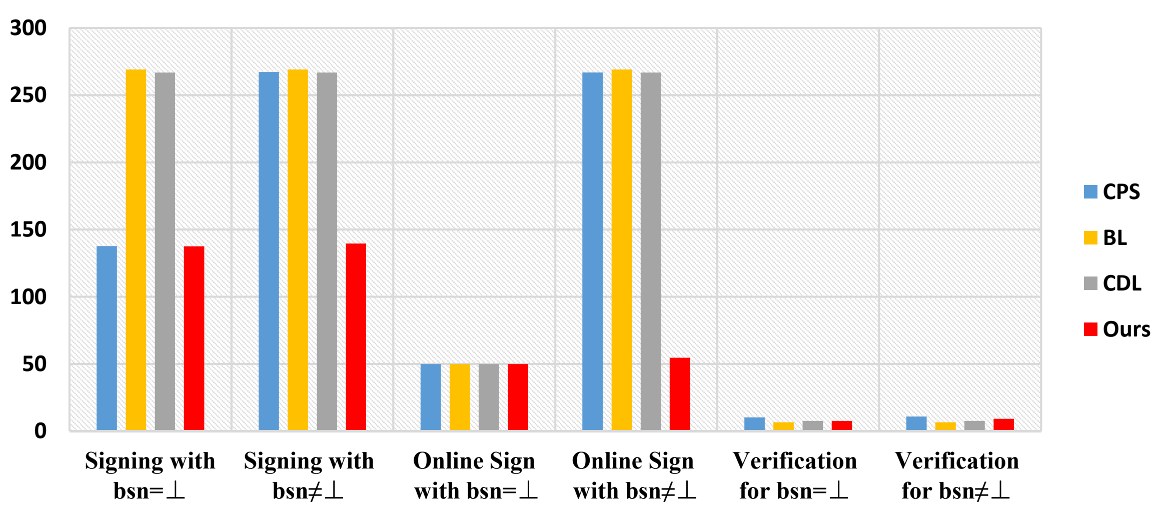

Fig. 1. Efficiency Comparison of DAA Schemes Supported by TPM 2.0 (in millisecond)

We propose the first DAA scheme with fully optimal TPM signing efficiency and denote it by $\mathtt{DAA}_{\mathtt{OPT}}$. That is, $\mathtt{DAA}_{\mathtt{OPT}}$ allows the TPM to cost only a single exponentiation in a prime-order group $\mathbb{G}_1$ when creating a signature for both signature modes. Moreover, the single exponentiation can be pre-computed, which allows our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ to obtain fast on-line signing time. Additionally, we present a simple implementation trick of parallel computation to reduce the signing time consuming at the host side. Our ECDAA scheme $\mathtt{DAA}_{\mathtt{OPT}}$ is provably secure under the DDH, DBDH and q-SDH assumptions in the Universally Composable (UC) security model [CDL16b,CDL16a] and the random oracle model [BR93].

Our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ is compatible with the TPM 2.0 specification, i.e., $\mathtt{DAA}_{\mathtt{OPT}}$ can be implemented using the existing TPM 2.0 commands. At the first time, we consider the TPM 2.0 implementations of three important DAA use cases, i.e., quoting PCR values, certifying a TPM key, and signing an arbitrary message given by the host, where these DAA use cases are corresponding to three types of DAA applications.

We have implemented and benchmarked several TPM 2.0 commands on an Infineon TPM 2.0 chip with vendor ID IFXSLB9670, which is installed on a module designed for the Raspberry Pi. Our benchmark results allow one to evaluate the TPM signing performance for three DAA use cases. Based on the benchmark results for the TPM 2.0 commands along with the known benchmark results [BCN14] on a labtop with 2.9GHz Intel Core i7-3520M CPU over the pairing-friendly Barreto-Naehrig (BN) curve [BN06], we compare our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ with the known DAA schemes [CPS10] (denoted by `CPS`), [BL10b] (denoted by `BL`), [CDL16a] (denoted by `CDL`) supported by TPM 2.0 in Figure 1 in terms of the total signing efficiency, online signing efficiency and verification efficiency. In particular, our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ has a factor 1.9x faster total signing time and 4.9x faster on-line signing time than these compared DAA schemes when creating a pseudonymous signature. When generating a fully anonymous signature, $\mathtt{DAA}_{\mathtt{OPT}}$ is a factor 1.9x more efficient than `BL` and `CDL`, and has the same efficiency as `CPS`. Our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ is more efficient than `CPS` in terms of the verification efficiency.

When applying our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ with a fully anonymous signature mode into the FIDO authentication framework, we can solve the problem of sharing the attestation key and reduce the signing cost of the TPM from three exponentiations in FIDO ECDAA to one exponentiation. We can also apply $\mathtt{DAA}_{\mathtt{OPT}}$ to mobile devices with ARM TrustZone by using fTPM to perform the TPM operations, and provide the advantage of better on-line signing performance compared to known DAA schemes supported by TPM 2.0.

In addition, our DAA scheme $\mathtt{DAA}_{\mathtt{OPT}}$ supports selective attribute disclosure, which can satisfy more application requirements. We also extend $\mathtt{DAA}_{\mathtt{OPT}}$ to support signature-based revocation and to guarantee privacy in presence of subverted TPMs. The two extended DAA schemes keep the TPM signing efficiency fully optimal, and provide significantly better signing performance than known related schemes.

1.2 Related Work

LRSW-DAA: Brickell et al. [BCL08] proposed the first LRSW-DAA scheme over symmetric bilinear groups. This scheme is further improved in [CMS08,CPS10] over asymmetric bilinear groups. Later, Bernhard et al. [BFG+13] utilized the special algebraic structure of randomized credentials, which implicitly contain unlinkable tags, to minimize the TPM's signing overhead for fully anonymous signature mode. However, their LRSW-DAA scheme still requires three exponentiations in $\mathbb{G}_1$ for the TPM to create a pseudonymous signature. Brickell et al. [BCL12] uses the batch proof and verification technique to construct the most efficient LRSW-DAA scheme for now, which reduces the signing overhead of the TPM to two exponentiations in $\mathbb{G}_1$ for the generation of a pseudonymous signature. However, this scheme is not compatible with the TPM 2.0 specification [Tru16]. Canard et al. [CPS14] proposed an efficient approach to delegate the computations of the TPM to the host in the interactive zero-knowledge proofs of knowledge. Using their method to the proofs of knowledge for pseudonymous signatures in Bernhard et al.'s DAA scheme [BFG+13], they show that the TPM could pre-compute one exponentiation in $\mathbb{G}_2$, and compute on-line one exponentiation in $\mathbb{G}_1$. Although the signing efficiency of the TPM is not improved, the TPM's on-line signing cost is reduced by two exponentiations in group $\mathbb{G}_1$. However, the group operations in $\mathbb{G}_2$ for the TPM are not supported by TPM 2.0. All the above LRSW-DAA schemes do not support attributes.

SDH-DAA: Chen and Feng [CF08] introduced the first SDH-DAA scheme. Chen [Che10] improved the signing efficiency of the TPM via removing an element of the credential. Later, Brickell and Li [BL10b] further improved the signing efficiency of the TPM by changing the way of delegation computation between the TPM and host, such that the TPM takes three exponentiations in $\mathbb{G}_1$ per sign protocol run. Recently, Camenisch et al. [CDL16a] proposed an efficient proof of knowledge for BBS+ signatures [ASM06], and then constructed a SDH-DAA scheme which improves the signing efficiency on the host side. Their scheme is the most efficient SDH-DAA scheme for now, but still requires three exponentiations in $\mathbb{G}_1$ for the TPM to generate a signature for both modes of signatures.

Signature-Based Revocation: Brickell and Li [BL07,BL10a] introduced Enhanced Privacy ID (EPID), which extends DAA with signature-based revocation. This revocation extension allows one to revoke a platform, based on a previous signature from the platform, even if the signature is fully anonymous. While private key revocation in DAA allows to revoke a platform by adding the platform's secret key to the revocation list, signature-based revocation allows for revocation without knowing the secret key of the platform and is an improvement over private key revocation. The pairing-based EPID scheme [BL10a] is recommended by Intel to serve as the industry standard for privacy-preserving authentication in Internet of Things (IoTs). These EPID schemes [BL07,BL10a] require $6n_r$ exponentiations for the TPM to prove that the platform has not been revoked, where n_r is the size of the signature revocation list. This is too expensive for a TPM with limited resources. Recently, Camenisch et al. [CDL16a] showed how to delegate the TPM's partial computations to the host in the signature-based revocation, which reduces the overhead of the TPM to $3n_r$ exponentiations. However, it is still too expensive for the TPM with limited resources.

DAA with Subverted TPMs: Recently, Camenisch et al. [CDL17] considered the setting that TPMs are possible to be subverted, i.e., the TPMs are created by a compromised manufacturer. A subverted TPM may create a subliminal channel (i.e., embedding some information into a signature) to compromise the privacy of a user. They proposed a DAA scheme with subverted TPMs, which requires two (resp., one) exponentiations for the TPM to produce a pseudonymous (resp., fully anonymous) signature. However, their DAA scheme requires that the TPM performs group operations in $\mathbb{G}_2$, and thus cannot be implemented by the TPM 2.0 commands, even if one can modify the TPM 2.0 commands with small changes. Later, Camenisch et al. (S&P'17) [CCD+17] modified the TPM 2.0 commands with minimal changes to the current TPM 2.0 commands, and showed that the modified TPM 2.0 commands can avoid a subliminal channel. Then, they used the modified TPM 2.0 commands to implement two ECDAA schemes with subverted TPMs, where one is based on the q-SDH assumption [BB08] and the other is based on a generalized variant of the LRSW assumption. Both the two schemes support signature-based revocation, but only the q-SDH-based scheme considers the support of attributes. Their signature-based revocation mechanism [CCD+17] still requires $3n_r$ exponentiations for the TPM when proving the platform has not been revoked.

1.3 Organization

We present the preliminaries in Section 2. We recall the definitions of DAA schemes in Section 3. In Section 4, we present the construction of our DAA scheme $\mathtt{DAA}_{\mathtt{OPT}}$ and two ways to further improve the efficiency of $\mathtt{DAA}_{\mathtt{OPT}}$, and also give an informal security analysis of $\mathtt{DAA}_{\mathtt{OPT}}$. In Section 5, we present the TPM 2.0 implementation of our DAA scheme involving three use cases of DAA. We evaluate the performance of our DAA scheme via comparing it with known DAA schemes supported by TPM 2.0 in Section 6. Signature-based revocation extension of our DAA scheme is shown in Appendix D, and we extend our DAA scheme to guarantee privacy against subverted TPMs in Appendix E. We provide an alternative description of our DAA scheme for UC security in Appendix B, and give a full formal security proof in Appendix C.

2 Preliminaries

Throughout this paper, we denote the security parameter by λ. We use $x \xleftarrow{\$} S$ to denote that sampling x uniformly at random from a set S. For a group $\mathbb{G}$, $\mathbb{G}^*$ denotes the set $\mathbb{G}\backslash\{1_{\mathbb{G}}\}$, where $1_{\mathbb{G}}$ is the identity element of $\mathbb{G}$. We use $[n]$ to denote the set $\{1, \dots, n\}$. We say that a function $f : \mathbb{N} \rightarrow [0, 1]$ is *negligible* if for every positive polynomial $\mathsf{poly}(\cdot)$ and all sufficiently large λ such that $f(\lambda) < 1/\mathsf{poly}(\lambda)$. We say that a function f is *overwhelming* if $1 - f$ is negligible.

Below, we recall the definition of bilinear groups. Then, we roughly review the signature proofs of knowledge. Next, we recall the DBDH, DDH and q-SDH assumptions.

2.1 Bilinear Groups

Let $\mathcal{G}$ be a *probabilistic polynomial time* (PPT) bilinear-group generator that on input a security parameter 1^λ, outputs a bilinear group $\Lambda = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order p, g_1 and g_2 are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map.

We say that $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map (pairing) if it is efficiently computable and satisfies the following properties: 1) bilinearity, i.e., $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}\ \forall a, b \in \mathbb{Z}_p$; 2) non-degeneracy, i.e., $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$ for all generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Following [GPS08], pairings are categorized into three types: 1) Type-1 pairings (a.k.a. symmetric pairings) have $\mathbb{G}_1 = \mathbb{G}_2$; 2) Type-2 pairings require $\mathbb{G}_1 \neq \mathbb{G}_2$, but there exists an efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $g_1 = \psi(g_2)$; 3) Type-3 pairings provide $\mathbb{G}_1 \neq \mathbb{G}_2$, but now there is no efficiently computable isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$. Type-2 and Type-3 pairings are called asymmetric pairings. Throughout this paper, we only consider Type-3 pairings.

2.2 Signature Proofs of Knowledge

We will use the notation introduced by Camenisch and Stadler [CS97] to abstract Signature Proofs of Knowledge (SPKs) on proving knowledge of discrete logarithms and statements about them. The SPKs can be obtained using Fiat-Shamir heuristic [FS86] to transform the corresponding Sigma protocols. For instance, $\pi \leftarrow \mathsf{SPK}\{(x) : y = g^x\}(m)$ denotes a signature proof of knowledge π on a message m, which proves knowledge of a witness x such that $y = g^x$, where $\mathbb{G} = \langle g \rangle$ is a group of prime order p. The SPKs are zero-knowledge via programming the random oracle and knowledge extractable in the random oracle model [PS00].

2.3 Assumptions

Assumption 1 (DBDH)**.** We say that the Decisional Bilinear Diffie-Hellman (DBDH) assumption [BB04] holds for $\mathcal{G}$ if any PPT adversary $\mathcal{A}$ and $\Lambda = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T,\ e, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$, there exists a negligible function $\nu(\cdot)$ such that

$$\begin{aligned}&\big|\Pr[a, b, c \xleftarrow{\$} \mathbb{Z}_p : \mathcal{A}(\Lambda, g_1^a, g_2^b, g_1^c, g_2^c, e(g_1, g_2)^{abc}) = 1] -\\ &\Pr[a, b, c, d \xleftarrow{\$} \mathbb{Z}_p : \mathcal{A}(\Lambda, g_1^a, g_2^b, g_1^c, g_2^c, e(g_1, g_2)^d) = 1]\big| \leq \nu(\lambda).\end{aligned}$$

In fact, the above assumption is an asymmetric version of the original DBDH assumption [BB04] for symmetric bilinear pairings. Recently, Desmoulins et al. [DLST14] used an analogous asymmetric version of the original DBDH assumption, where the adversary is given an additional element g_1^b as input. Freire et al. [FHKP13] used an asymmetric version of the original DBDH assumption over Type-2 pairings (DBDH-2) as introduced in [Gal05], where the adversary is given $(g_2, g_1^a, g_2^b, g_2^c)$ as input. For Type-2 pairings, the elements g_1^b and g_1^c can be computed via $\psi(g_2^b)$ and $\psi(g_2^c)$ respectively. Thus, the adversary is actually given $(g_1, g_2, g_1^a, g_1^b, g_2^b, g_1^c, g_2^c)$ as input in the DBDH-2 assumption.

Assumption 2 ($\mathrm{DDH}_{\mathbb{G}_1}$)**.** We say that the Decisional Diffie-Hellman (DDH) assumption holds in group $\mathbb{G}_1$ if for any PPT adversary $\mathcal{A}$ and $\Lambda = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$, there exists a negligible function $\nu(\cdot)$ such that

$$\big|\Pr[a, b \xleftarrow{\$} \mathbb{Z}_p : \mathcal{A}(\Lambda, g_1^a, g_1^b, g_1^{ab}) = 1] - \Pr[a, b, c \xleftarrow{\$} \mathbb{Z}_p : \mathcal{A}(\Lambda, g_1^a, g_1^b, g_1^c) = 1]\big| \leq \nu(\lambda).$$

Assumption 3 (q-SDH)**.** We say that the q-Strong Diffie-Hellman (q-SDH) assumption [BB08] holds for $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$ and $\Lambda = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$, there exists a negligible function $\nu(\cdot)$ such that

$$\Pr[\gamma \xleftarrow{\$} \mathbb{Z}_p^* : (g_1^{1/(\gamma+c)}, c) \leftarrow \mathcal{A}(\Lambda, g_1^\gamma, \ldots, g_1^{\gamma^q}, g_2^\gamma)] \leq \nu(\lambda),$$

where $c \in \mathbb{Z}_p \backslash \{-\gamma\}$.

3 Definitions of DAA Schemes

In this section, we review the syntax of DAA schemes and the desired security properties for DAA, i.e., *anonymity*, *unforgeability* and *non-frameability*. We adopt the security model for DAA by Camenisch et al. [CDL16b], which is defined as an ideal functionality $\mathcal{F}_{\mathsf{daa}}^{l}$ in the Universal Composability (UC) framework [Can01]. We extend this model to support the functionality of attributes by following the extension [CDL16a]. We refer the reader to Appendix A (or [CDL16b,CDL16a]) for the formal security definition of DAA in the form of an ideal functionality.

3.1 Syntax of DAA Schemes

In a DAA scheme, there are four types of parties: TPM $\mathcal{M}_i$ and host $\mathcal{H}_j$ constituting a platform, issuer $\mathcal{I}$ and verifier $\mathcal{V}$. The DAA scheme consists of three algorithms **Setup**, **Verify** and **Link**, and two protocols **Join** and **Sign**.

Setup. Given a set of system parameters params on a security parameter λ, an issuer $\mathcal{I}$ generates its public key ipk and secret key isk, where params and ipk are publicly available. We assume that params and ipk are implicit inputs for the following protocols and algorithms.

Join. This is an interactive protocol between a platform $(\mathcal{M}_i, \mathcal{H}_j)$ and the issuer $\mathcal{I}$ who decides whether the platform is allowed to become a member. By executing the join protocol, the platform creates a secret key gsk, and receives a number of attributes $attrs = (a_1, \ldots, a_n)$ and a credential cre given by $\mathcal{I}$. The credential cre certifies the secret key gsk and attributes $attrs$, where the attributes include more information about the platform such as the manufacturer and model version and an expiration date of the credential etc.

Sign. After being a member, a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$ can jointly sign a message m w.r.t. a basename bsn resulting in a signature σ, where bsn is either the name string of a verifier or a special symbol $\perp$. We refer to σ as a fully anonymous signature if $\mathsf{bsn} = \perp$ and a pseudonymous signature otherwise. The platform can also selectively disclose a part of attributes from its credential cre, e.g., disclosing that the signature was created by a TPM of a certain manufacturer or the expiration date of the credential. We denote the disclosure of attributes by (D, I), where $D \subseteq \{1, \ldots, n\}$ is a set indicating which attributes are disclosed, $I = (a_1, \ldots, a_n)$ is a tuple specifying the disclosed attribute values, and a_i is set as $\perp$ if the i-th attribute is not disclosed. We also denote by $\bar{D}$ the set of the indices of undisclosed attributes, i.e., $\bar{D} = \{1, \ldots, n\} \backslash D$.

Verify. Given a message m, a basename bsn, a signature σ, an attribute disclosure (D, I) and a revocation list $\mathtt{RL}$ consisting of the secret keys of corrupted platforms, a verifier $\mathcal{V}$ can run a *deterministic* verification algorithm to check that σ is valid on m w.r.t. bsn and stems from a platform that holds a credential satisfying the predicate defined by (D, I). The verification algorithm outputs 1 if the check passes and 0 otherwise.

The revocation list $\mathtt{RL}$ is used to support private key revocation. When a secret key (private key) of a corrupted platform is exposed, the secret key would be added to $\mathtt{RL}$, which allows a verifier to recognize and thus reject all the signatures created by the secret key.

Link. On input two message/signature pairs (m_0, σ_0) and (m_1, σ_1), attribute disclosure (D_0, I_0) and (D_1, I_1) and a basename $\mathsf{bsn} \neq \perp$, a verifier $\mathcal{V}$ can run a *deterministic* link algorithm to decide whether the two signatures link or not. If both σ_0 and σ_1 are valid on respective $(m_0, (D_0, I_0))$ and $(m_1, (D_1, I_1))$ w.r.t. the same $\mathsf{bsn} \neq \perp$ and were produced by the same secret key, the link algorithm outputs 1 (linked). Otherwise, the link algorithm outputs $\perp$ if one of σ_0 and σ_1 is invalid and 0 (unlinked) otherwise.

3.2 Desired Security Properties for DAA

Following [CDL16b], a DAA scheme should satisfy the following desired security properties:

Anonymity. Given two signatures with respect to different basenames or $\mathsf{bsn} = \perp$, no adversary can distinguish whether both signatures were generated by the same honest platform, or whether they were created by two different honest platforms. The property requires that the entire platform (TPM+host) is honest, and should hold even if the issuer is corrupted.

Unforgeability. This property requires that the issuer is honest, and should hold even if some or all hosts are corrupted.

1. If all unrevoked TPMs are honest, no adversary can produce a signature on a message m w.r.t. a basename bsn and attribute disclosure (D, I), when no platform that joined with those attributes signed m w.r.t. bsn and (D, I).
2. An adversary can only sign in the name of corrupted TPMs. More precisely, if k corrupted and unrevoked TPMs joined with attributes fulfilling attribute disclosure (D, I) for some integer k, the adversary can create at most k unlinkable signatures w.r.t. the same basename $\mathsf{bsn} \neq \perp$ and attribute disclosure (D, I).

Non-frameability. No adversary can create a signature on a message m w.r.t. a basename bsn which links to a signature created by an honest platform, when the platform never signed m w.r.t. bsn. The property requires that the entire platform is honest, and should hold even if the issuer is corrupted.

4 Our DAA Scheme

We present the construction of our DAA scheme (denoted by $\mathtt{DAA_{OPT}}$). Our scheme $\mathtt{DAA_{OPT}}$ supports selective attribute disclosure, and would be degraded as the traditional DAA scheme when removing the attributes (i.e., $n = 0$). Following [CDL16a], we consider that only the secret key is protected by the TPM and all attributes are stored on the host in order to obtain better efficiency. We will further improve the computational efficiency of $\mathtt{DAA_{OPT}}$ by presenting online/offline DAA signatures and a simple implementation trick of parallel computation. We prove that the protocol $\mathtt{DAA_{OPT}}$ securely realizes the ideal functionality $\mathcal{F}^{l}_{\mathsf{daa}}$ with static corruption and attributes defined in [CDL16b,CDL16a] under the DBDH, $\mathrm{DDH}_{\mathbb{G}_1}$ and q-SDH assumptions in the random oracle model, based on the proofs by Camenisch et al. [CDL16b,CDL16a]. We informally argue the security of $\mathtt{DAA_{OPT}}$ in this section, and give the detailed security proof in Appendix C. First of all, we describe the ideas underlying the construction of $\mathtt{DAA_{OPT}}$.

4.1 High Level Description

We adopt the BBS+ signature to issue DAA credentials, where the BBS+ signature scheme was proposed in [ASM06] based on the schemes [BBS04,CL04]. This means that a platform consisting of a TPM and a host will obtain a credential (A, x, u) on a secret key gsk and attributes $attrs = (a_1, \ldots, a_n)$ such that $A = (g_1 \bar{g}^{gsk} h_0^u \prod_{i=1}^{n} h_i^{a_i})^{1/(\gamma + x)}$ in the join protocol, where n is the number of attributes and γ is the secret key of the issuer. We use the proof of knowledge for BBS+ signatures in the full version of [CDL16a] to prove possession of such a credential.[4] In particular, the credential A is randomized as T_1 and the validity of T_1 is proved using a signature proof of knowledge.

Except for the proof of knowledge of a credential, a *pseudonym* and its proof are included in a DAA signature for $\mathsf{bsn} \neq \perp$. Furthermore, an *unlinkable tag* and its proof are also involved in a signature to support private-key revocation for $\mathsf{bsn} = \perp$. In the existing DAA schemes, a pseudonym is set as $K = H_{\mathbb{G}}(\mathsf{bsn})^{gsk}$, and an unlinkable tag is set as $(B, K = B^{gsk})$ for a random $B \in \mathbb{G}$ or $B = H_{\mathbb{G}}(\mathsf{str})$ with a random string str, where $H_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ is a random oracle, $\mathbb{G}$ is a cyclic group such as $\mathbb{G}_1$ and gsk is the TPM's secret key. This results in that the TPM needs to cost two exponentiations to compute K and prove the validity of K in known DAA schemes. In this paper, we propose two techniques to achieve the fully optimal TPM signing efficiency.

For pseudonymous signature mode, we propose a technique of delegable pseudonyms, which is inspired by Canard et al.'s method [CPS14] on delegation of zero-knowledge proofs of knowledge. Specifically, a pseudonym on a basename bsn is computed as $K = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^{gsk}$, where $H_{\mathbb{G}_2} : \{0,1\}^* \to \mathbb{G}_2$ is a random oracle. The new construction of pseudonyms allows the TPM to delegate the computations of a pseudonym K and a commitment $L = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^r$ to the host, where L is used to prove knowledge of gsk such that $K = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^{gsk}$ and r is chosen at random by the TPM. Concretely, the host stores a public key $gpk = \bar{g}^{gsk}$ created by the TPM in the join protocol and receives a commitment $E = \bar{g}^r$ from the TPM in

[4] This proof of knowledge is based on the proofs of knowledge for the weak Boneh-Boyen signature in [ALT+15,CDH16]. Concurrently, Barki et al. [BBDT17] gave a similar proof of knowledge for BBS+ signatures but slightly less efficient.

the sign protocol. Then, the host can compute K and L via $K \leftarrow e(gpk, H_{\mathbb{G}_2}(\mathsf{bsn}))$ and $L \leftarrow e(E, H_{\mathbb{G}_2}(\mathsf{bsn}))$ respectively.

From the construction of pseudonyms, we can see that $gpk = \bar{g}^{gsk}$ must keep hidden, and otherwise gpk can be used to identify the signatures. Thus, the platform cannot directly send gpk to the issuer in the join protocol. A possible way is to let the platform send a Pedersen commitment $C = \bar{g}^{gsk} h_0^{u'}$ to the issuer. However, this way is not compatible with the TPM 2.0 specification. That is, by the existing TPM 2.0 commands, neither the TPM could create a Pedersen commitment C nor the TPM could check whether the commitment C' received by the issuer is created correctly using the TPM public key gpk when the host chooses the randomness u'. To be compatible with the TPM 2.0 specification, we split the key gsk into a secret key tsk chosen by the TPM and a secret key hsk picked by the host via $gsk \leftarrow tsk + hsk$, where the technique of splitting keys was previously used for guaranteeing privacy against subverted TPMs in [CDL17]. Specifically, the TPM sends a public key $tpk = \bar{g}^{tsk}$ to the host, and the host stores $gpk = tpk \cdot \bar{g}^{hsk}$. In the join protocol, the host picks $u' \xleftarrow{\$} \mathbb{Z}_p$ and computes $C \leftarrow \bar{g}^{hsk} \cdot h_0^{u'}$, and then sends tpk and C to the issuer for requesting a credential. Now, a commitment L can be computed by the host via picking $\hat{r} \xleftarrow{\$} \mathbb{Z}_p$ and computing $L \leftarrow e(E \cdot \bar{g}^{\hat{r}}, H_{\mathbb{G}_2}(\mathsf{bsn}))$.

For fully anonymous signature mode, we present a technique of delegable unlinkable tags to delegate the computations of an unlinkable tag $(B, K = B^{gsk})$ and the corresponding commitment $L = B^{r+\hat{r}}$ to the host, where $r, \hat{r}$ are chosen at random by the TPM and host respectively. Specifically, the host picks $b \xleftarrow{\$} \mathbb{Z}_p^*$, and then can compute (B, K) (resp., L) via randomizing $(\bar{g}, gpk)$ (resp., E) as $(B = \bar{g}^b, K = gpk^b)$ (resp., $L = (E \cdot \bar{g}^{\hat{r}})^b$).

4.2 Detailed Construction

We assume the public availability of system parameters $\mathsf{params} = (\lambda, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \bar{g}, \ell_n)$, where λ is a security parameter, $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ is a set of bilinear group parameters generated by $\mathcal{G}(1^\lambda)$, $\bar{g} \in \mathbb{G}_1$ is a fixed generator and ℓ_n denotes the bit length of nonce picked by TPMs. We will use four independent hash functions $H_i : \{0,1\}^* \rightarrow \mathbb{Z}_p$ for $\forall i \in \{1,2,3\}$ and $H_{\mathbb{G}_2} : \{0,1\}^* \rightarrow \mathbb{G}_2$ modeled as random oracles. Note that $H_{\mathbb{G}_2}$ can be implemented fast using the hashing algorithms [FCKRH12,BP17], and the speed of calculating $H_{\mathbb{G}_2}$ is doubled in the case of BN curves [FCKRH12].

Setup. Given the system parameters params, an issuer $\mathcal{I}$ creates its public/private key pair $(\mathsf{ipk}, \mathsf{isk})$ as follows:

1. Choose $h_0, h_1, \ldots, h_n \xleftarrow{\$} \mathbb{G}_1^*$.
2. Pick $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$, and compute $w \leftarrow g_2^\gamma$.
3. Prove knowledge of secret key γ on public key w by

$$\pi_1 \leftarrow \mathsf{SPK}_1\{(\gamma) : w = g_2^\gamma\}(\text{“setup”}).$$

4. Set $\mathsf{ipk} \leftarrow (\{h_i\}_{i=0}^n, w, \pi_1)$ and $\mathsf{isk} \leftarrow \gamma$.

SPK_1 can be constructed in the following standard way: 1) pick $r \xleftarrow{\$} \mathbb{Z}_p$ and compute a *commitment* $R \leftarrow g_2^r$; 2) generate a *challenge* $c \leftarrow H_3(\text{“setup”}, g_2, w, R)$; 3) produce a *response* $s \leftarrow r + c \cdot \gamma \mod p$; 4) output a proof $\pi_1 \leftarrow (c, s)$. The proof $\pi_1 = (c, s)$ can be easily verified publicly by doing the following: 1) recover a commitment $R' \leftarrow g_2^s \cdot w^{-c}$; 2) compute $c' \leftarrow H_3(\text{“setup”}, g_2, w, R')$; 3) accept the proof if $c = c'$ and reject it otherwise. The issuer $\mathcal{I}$ also registers its public key ipk at a Certification Authority (CA) such that anyone can get the public key ipk correctly.

Join. The join protocol executed between the TPM $\mathcal{M}_i$, host $\mathcal{H}_j$ and issuer $\mathcal{I}$ is shown in Figure 2, where `JOIN` denotes a join request. We assume that $\mathcal{M}_i$ can authenticate itself to $\mathcal{I}$ and convince $\mathcal{I}$ that tpk is created by a legitimate TPM. This can be realized by enabling $\mathcal{M}_i$ and $\mathcal{I}$ to communicate over a semi-authenticated channel, meaning that a message sent to the issuer consists of an authenticated part (i.e., tpk) and an unauthenticated part (i.e., (C, π_t, π_h)). Multiple methods can be adopted to establish the semi-authenticated channel using the TPM's endorsement key [Tru16], where an overview is provided in [BFG$^+$13]. We can adopt the method in [CW10] to establish the semi-authenticated channel, where the method has been adopted by the TCG in the TPM 2.0 specification [CL13,Tru16]. Besides, by using this method [CW10], $\mathcal{I}$ can send a

TPM $\mathcal{M}_i$		Host $\mathcal{H}_j$		Issuer $\mathcal{I}$ ($\mathsf{isk} = \gamma$)
		$hsk, u' \xleftarrow{\$} \mathbb{Z}_p$,		
$tsk \xleftarrow{\$} \mathbb{Z}_p$	$\xleftarrow{\texttt{TPM.Create}}$	$hpk \leftarrow \bar{g}^{hsk}$	$\xrightarrow{\texttt{JOIN}}$	
$tpk \leftarrow \bar{g}^{tsk}$		$C \leftarrow hpk \cdot h_0^{u'}$		$N_I \xleftarrow{\$} \{0,1\}^{\lambda}$
	$\xrightarrow{tpk}$	$gpk \leftarrow tpk \cdot hpk$	$\xleftarrow{N_I}$	
$r \xleftarrow{\$} \mathbb{Z}_p$	$\xleftarrow{\texttt{TPM.Commit}}$	$\hat{r}, r' \xleftarrow{\$} \mathbb{Z}_p,\ R \leftarrow \bar{g}^{\hat{r}} \cdot h_0^{r'}$		
$E \leftarrow \bar{g}^{r}$	$\xrightarrow{E}$	$c_h \leftarrow H_2(\text{“TPM.join”}, \bar{g}, tpk, E, N_I)$		
$N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$	$\xleftarrow{\texttt{TPM.Sign}, c_h}$	$z \leftarrow H_2(\text{“Host.join”}, \bar{g}, h_0, C, R, N_I)$		
$c \leftarrow H_1(N_t, c_h)$		$\hat{s} \leftarrow \hat{r} + z \cdot hsk \mod p$		
$s \leftarrow r + c \cdot tsk \mod p$		$s' \leftarrow r' + z \cdot u' \mod p$		
	$\xrightarrow{(N_t, s)}$	$c \leftarrow H_1(N_t, c_h)$		
		$\pi_t \leftarrow (c, s, N_t)$		
		$\pi_h \leftarrow (z, \hat{s}, s')$	$\xrightarrow{(tpk, C, \pi_t, \pi_h)}$	
				Parse π_t as (c, s, N_t)
				Parse π_h as $(z, \hat{s}, s')$
				$E' \leftarrow \bar{g}^{s} \cdot tpk^{-c}$
				$R' \leftarrow \bar{g}^{\hat{s}} \cdot h_0^{s'} \cdot C^{-z}$
				$c_h' \leftarrow H_2(\text{“TPM.join”}, \bar{g}, tpk, E', N_I)$
				$c' \leftarrow H_1(N_t, c_h')$
				$z' \leftarrow H_2(\text{“Host.join”}, \bar{g}, h_0, C, R', N_I)$
				If $c' \neq c$ or $z' \neq z$, abort.
				$u'', x \xleftarrow{\$} \mathbb{Z}_p$
				$A \leftarrow (g_1 \cdot tpk \cdot C \cdot h_0^{u''} \cdot \prod_{i=1}^{n} h_i^{a_i})^{1/(\gamma+x)}$
				$attrs \leftarrow (a_1, a_2, \ldots, a_n)$
		$u \leftarrow u' + u'' \mod p$	$\xleftarrow{(A, x, u'', attrs)}$	
		$Y \leftarrow g_1 \cdot gpk \cdot h_0^{u} \cdot \prod_{i=1}^{n} h_i^{a_i}$		
		If $e(A, w \cdot g_2^{x}) \neq e(Y, g_2)$, abort.		
		Store $cre \leftarrow (A, x, u, Y, gpk, hsk)$ and $attrs$		

Fig. 2. The join protocol of $\mathtt{DAA}_{\mathtt{OPT}}$. The notation `TPM.Create`, `TPM.Commit` and `TPM.Sign` represent the TPM requests of the following procedures: *creating a TPM key*, *generating a commitment* and *generating a signature* respectively. Note that they are not real TPM 2.0 commands.

credential (A, x, u'') and the attributes $attrs$ to the platform in a confidential manner via encrypting them with an encryption scheme.

In the join protocol, $\mathcal{M}_i$ creates a public key tpk and $\mathcal{H}_j$ produces a Pedersen commitment [Ped92] $C = \bar{g}^{hsk} h_0^{u'}$. Then, $\mathcal{M}_i$ proves knowledge of secret key tsk with the help of $\mathcal{H}_j$, i.e., they cooperatively produce a signature proof of knowledge

$$\pi_t \leftarrow \mathsf{SPK}_t\{(tsk) : tpk = \bar{g}^{tsk}\}(\text{“TPM.join”}, N_I).$$

The host $\mathcal{H}_j$ also proves knowledge of secret key hsk and randomness u' via independently generating

$$\pi_h \leftarrow \mathsf{SPK}_h\{(hsk, u') : C = \bar{g}^{hsk} h_0^{u'}\}(\text{“Host.join”}, N_I).$$

Upon receiving a tuple (tpk, C, π_t, π_h), $\mathcal{I}$ checks the validity of proofs π_t and π_h, and then blindly issues a BBS+ signature (A, x, u'') on key gsk and attributes $attrs = \{a_i\}_{i=1}^{n}$ to platform $(\mathcal{M}_i, \mathcal{H}_j)$, where $gsk =$

TPM $\mathcal{M}_i$ (tsk)		Host $\mathcal{H}_j$ $(cre, m, \mathsf{bsn}, D, I)$
		$t_1 \xleftarrow{\$} \mathbb{Z}_p^*$, $t_2 \xleftarrow{\$} \mathbb{Z}_p$
	$\xleftarrow{\texttt{TPM.Commit}}$	$t_3 \leftarrow 1/t_1 \mod p$, $\tilde{u} \leftarrow u - t_2 \cdot t_3 \mod p$
$r \xleftarrow{\$} \mathbb{Z}_p$		$T_1 \leftarrow A^{t_1}$, $T_2 \leftarrow Y^{t_1} \cdot T_1^{-x}$, $Y' \leftarrow Y^{t_1} \cdot h_0^{-t_2}$
$E \leftarrow \bar{g}^r$		$\hat{r}, r_x, r_{\tilde{u}}, r_{t_2}, r_{t_3} \xleftarrow{\$} \mathbb{Z}_p$, $r_{a_i} \xleftarrow{\$} \mathbb{Z}_p$ for each $i \in \bar{D}$,
	$\xrightarrow{E}$	$\hat{E} \leftarrow \bar{g}^{\hat{r}}$, $\tilde{R}_1 \leftarrow Y'^{-r_{t_3}} \cdot h_0^{r_{\tilde{u}}} \cdot \prod_{i \in \bar{D}} h_i^{r_{a_i}}$,
		$R_2 \leftarrow T_1^{-r_x} \cdot h_0^{r_{t_2}}$, $\tilde{E} \leftarrow E \cdot \hat{E}$, $R_1 \leftarrow \tilde{E} \cdot \tilde{R}_1$
		If $\mathsf{bsn} = \perp$, $b \xleftarrow{\$} \mathbb{Z}_p^*$, $B \leftarrow \bar{g}^b$, $K \leftarrow gpk^b$, $L \leftarrow \tilde{E}^b$.
		If $\mathsf{bsn} \neq \perp$, $B \leftarrow \perp$, $K \leftarrow e(gpk, H_{\mathbb{G}_2}(\mathsf{bsn}))$, $L \leftarrow e(\tilde{E}, H_{\mathbb{G}_2}(\mathsf{bsn}))$.
		$c_h \leftarrow H_2(\text{“sign”}, \bar{g}, g_1, \{h_i\}_{i=0}^n, T_1, T_2, Y', B, K, R_1, R_2, L)$
	$\xleftarrow{\texttt{TPM.Sign}, (m, \mathsf{bsn}, D, I, c_h)}$	
$N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$		
$c \leftarrow H_1(N_t, m, \mathsf{bsn}, D, I, c_h)$		
$s \leftarrow r + c \cdot tsk \mod p$		
	$\xrightarrow{(N_t, s)}$	$c \leftarrow H_1(N_t, m, \mathsf{bsn}, D, I, c_h)$
		$\bar{s} \leftarrow s + \hat{r} + c \cdot hsk \mod p$, $s_x \leftarrow r_x + c \cdot x \mod p$
		$s_{\tilde{u}} \leftarrow r_{\tilde{u}} + c \cdot \tilde{u} \mod p$, $s_{t_2} \leftarrow r_{t_2} + c \cdot t_2 \mod p$
		$s_{t_3} \leftarrow r_{t_3} + c \cdot t_3 \mod p$, $s_{a_i} \leftarrow r_{a_i} + c \cdot a_i \mod p$ for each $i \in \bar{D}$
		$\pi_2 \leftarrow (c, \bar{s}, s_x, s_{\tilde{u}}, s_{t_2}, s_{t_3}, \{s_{a_i}\}_{i \in \bar{D}}, N_t)$
		$\sigma \leftarrow (T_1, T_2, Y', B, K, \pi_2)$

Fig. 3. The sign protocol of $\texttt{DAA}_{\texttt{OPT}}$. For the case that $\mathsf{bsn} \neq \perp$, B is set as $\perp$, as $e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))$ can be computed offline by the verifier. The elements marked in the dashed box can be computed offline by the TPM and host. Again, `TPM.Commit` and `TPM.Sign` represent the TPM requests rather than real TPM 2.0 commands.

$tsk + hsk \mod p$ is the secret key of this platform. Except for the BBS+ signature (A, x, u) and secret key hsk, $\mathcal{H}_j$ stores $gpk = tpk \cdot \bar{g}^{hsk}$ and $Y = g_1 \cdot gpk \cdot h_0^u \cdot \prod_{i=1}^n h_i^{a_i}$ in credential cre for fast signing. To be compatible with TPM 2.0, the TPM does not output a digest c, and instead the host re-computes c from N_t and c_h.

Sign. A TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$ can cooperatively sign a message m w.r.t. basename bsn and attribute disclosure (D, I) by executing the sign protocol shown in Figure 3, where (D, I) is selectively disclosed by $\mathcal{H}_j$. To generate a signature, $\mathcal{H}_j$ randomizes A and Y as $T_1 = A^{t_1}$ and $Y' = Y^{t_1} h_0^{-t_2}$ respectively, and computes $T_2 = Y^{t_1} T_1^{-x}$ such that $T_2 = T_1^{\gamma}$. Then, $\mathcal{H}_j$ generates an unlinkable-tag/pseudonym $(B, K = B^{gsk})$, where either $B = \bar{g}^b$ or $B = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))$. Next, $\mathcal{M}_i$ cooperates with $\mathcal{H}_j$ to produce a signature proof of knowledge

$$\pi_2 \leftarrow \mathsf{SPK}_2\Big\{(gsk, \{a_i\}_{i \in \bar{D}}, x, \tilde{u}, t_2, t_3) : g_1^{-1} \prod_{i \in D} h_i^{-a_i} = Y'^{-t_3} \bar{g}^{gsk} h_0^{\tilde{u}} \prod_{i \in \bar{D}} h_i^{a_i} \ \wedge$$
$$T_2/Y' = T_1^{-x} h_0^{t_2} \ \wedge K = B^{gsk}\Big\}(\text{“sign”}, m, \mathsf{bsn}, D, I),$$

where $t_3 = t_1^{-1} \mod p$ and $\tilde{u} = u - t_2 t_3 \mod p$.

In the process of generating a proof π_2, host $\mathcal{H}_j$ calls TPM $\mathcal{M}_i$ to produce a signature proof of knowledge

$$\pi_t \leftarrow \mathsf{SPK}_t\{(tsk) : tpk = \bar{g}^{tsk}\}(\text{“sign”}, m, \mathsf{bsn}, D, I).$$

Verify. On input a message m, a basename bsn, a signature σ, attribute disclosure (D, I) and a revocation list $\texttt{RL}$, a verifier $\mathcal{V}$ can verify the signature as follows:

1. Parse signature σ as $(T_1, T_2, Y', B, K, \pi_2)$ and proof π_2 as $(c, \bar{s}, s_x, s_{\tilde{u}}, s_{t_2}, s_{t_3}, \{s_{a_i}\}_{i \in \bar{D}}, N_t)$.
2. Check that $B \neq 1_{\mathbb{G}_1}$ if $\mathsf{bsn} = \perp$ and $B = \perp$ otherwise. If $\mathsf{bsn} \neq \perp$, compute $B \leftarrow e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))$.
3. Check that $e(T_1, w) = e(T_2, g_2)$.

4. Verify the validity of proof π_2 as follows:
   (a) Compute the following three commitments:

$$R'_1 \leftarrow Y'^{-s_{t_3}} \cdot \bar{g}^{\bar{s}} \cdot h_0^{s_{\tilde{u}}} \cdot \prod_{i \in \bar{D}} h_i^{s_{a_i}} \cdot g_1^c \cdot \prod_{i \in D} h_i^{c \cdot a_i}$$
$$R'_2 \leftarrow T_1^{-s_x} \cdot h_0^{s_{t_2}} \cdot (T_2/Y')^{-c}$$
$$L' \leftarrow B^{\bar{s}} \cdot K^{-c}$$

   (b) Calculate $c'_h \leftarrow H_2(\text{"sign"}, \bar{g}, g_1, \{h_i\}_{i=0}^n, T_1, T_2, Y', B, K, R'_1, R'_2, L')$.
   (c) Compute $c' \leftarrow H_1(N_t, m, \mathsf{bsn}, D, I, c'_h)$.
   (d) Check that $c' = c$.
5. For every $gsk_i \in \mathtt{RL}$, check that $K \neq B^{gsk_i}$.
6. Output 1 if all the above checks pass and 0 otherwise.

Note that $e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))$ can be computed off-line by verifier $\mathcal{V}$. Besides, $\mathcal{V}$ can pre-compute $e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^{gsk_i}$ for each $gsk_i \in \mathtt{RL}$ and store the computational results for the verification of pseudonymous signatures.

Link. On input two message/signature pairs (m_0, σ_0) and (m_1, σ_1), attribute disclosure (D_0, I_0) and (D_1, I_1), and a basename $\mathsf{bsn} \neq \perp$, a verifier $\mathcal{V}$ decides if the two signatures link as follows:
1. Verify the validity of σ_0 and σ_1, i.e., output $\perp$ if the verification algorithm outputs 0 on input $(m_0, \mathsf{bsn}, \sigma_0, D_0, I_0, \mathtt{RL} = \emptyset)$ or $(m_1, \mathsf{bsn}, \sigma_1, D_1, I_1, \mathtt{RL} = \emptyset)$.
2. Parse σ_0 and σ_1 as $(T_{1,0}, T_{2,0}, Y'_0, B_0, K_0, \pi_{2,0})$ and $(T_{1,1}, T_{2,1}, Y'_1, B_1, K_1, \pi_{2,1})$ respectively.
3. If $K_0 = K_1$, output 1, otherwise output 0.

4.3 Efficiency Improvement

In this section, we present online/offline DAA signatures and a simple implementation trick of parallel computation to improve the computational efficiency of $\mathtt{DAA_{OPT}}$. We only describe the efficiency improvement of the sign protocol, but the two methods can be also applied to the join protocol.

Online/Offline DAA Signatures. The notion of online/offline signatures was introduced by Even, Goldreich and Micali [EGM96]. We apply the online/offline signing idea into $\mathtt{DAA_{OPT}}$ to obtain fast online signing time. In particular, we transform the sign protocol of $\mathtt{DAA_{OPT}}$ into an online/offline sign protocol, based on the fact that a basename bsn is submitted online by a verifier and a message m to be signed is determined online (e.g., m may include the PCR values of the current state of the host system or a nonce N_v from the verifier).

In the offline phase, TPM $\mathcal{M}_i$ can pre-compute a commitment $E = \bar{g}^r$, and host $\mathcal{H}_j$ can pre-compute the following elements: $T_1, T_2, Y', \tilde{E}, R_1, R_2, t_3, \tilde{u}$, as they are independent of m and bsn. That is, the elements marked in the dashed box of Figure 3 can be computed offline. In the online phase, $\mathcal{H}_j$ firstly computes B, K, L and c_h, and then $\mathcal{M}_i$ can generate (N_t, s) without any costly computation. Finally, $\mathcal{H}_j$ can rapidly complete the computation of a signature by re-computing a digest c and generating a proof $\pi_2 = (c, \bar{s}, s_x, s_{\tilde{u}}, s_{t_2}, s_{t_3}, \{s_{a_i}\}_{i \in \bar{D}}, N_t)$ fast. By default, $\mathcal{M}_i$ and $\mathcal{H}_j$ would securely delete the intermediate pre-computation results after the signatures are produced. For the case of $\mathsf{bsn} = \perp$, $\mathcal{H}_j$ could further pre-compute B, K, L and c_h, at the cost of differentiating that pre-computation results are used to create which type of signatures. In the above online/offline DAA signatures, we assume that the host is allowed to select offline the attribute disclosure (D, I). If some applications only allow that the attribute disclosure is determined online, then the host has to compute $\prod_{i \in \bar{D}} h_i^{r_{a_i}}$ and c_h online.

In a straightforward way, a randomness r is stored inside the TPM $\mathcal{M}_i$ after a `TPM.Commit` request and deleted after a `TPM.Sign` request. However, such implementation is too expensive for the TPM with limited storage, when multiple pre-computations are required. TPM 2.0 [Tru16] provides an alternative efficient implementation without storing the random numbers, which allows us to prepare the pre-computation values for multiple signatures. Roughly, the TPM generates a randomness r via a Key Derivation Function (KDF) with a secret seed and a counter, and maintains a bit table of fixed size to mark which random numbers have been used. We refer the reader to [Tru16,CL13] for the details.

Implementation Trick of Parallel Computation. The TPM is a small discrete chip with independent CPU and memory, and has much less resources than the host. Therefore, when TPM $\mathcal{M}_i$ is computing a commitment E, host $\mathcal{H}_j$ can compute in parallel the following elements: $t_3, \tilde{u}, T_1, T_2, Y', \hat{E}, \tilde{R}_1, R_2, B, K$, if the number $u = |\bar{D}|$ of undisclosed attributes is not very large. By using this implementation trick, the signing time consuming at the host side can be reduced significantly. This trick can be also applied to other DAA schemes. Although this implementation trick is simple, it has not been considered in all existing DAA schemes as best as we know.

4.4 Security Properties of Our DAA Scheme

In this section, we give an informal security analysis to argue the security of our protocol $\mathtt{DAA_{OPT}}$. For every security property as described in Section 3.2, we argue why $\mathtt{DAA_{OPT}}$ satisfies it. Note that this is structurally quite different from the actual security proof. In the actual proof, we prove that no environment $\mathcal{Z}$ can distinguish the real world where it is interacting with protocol $\mathtt{DAA_{OPT}}$ and adversary $\mathcal{A}$, from the ideal world where it is interacting with ideal functionality $\mathcal{F}^l_{\mathsf{daa}}$ and simulator $\mathcal{S}$. Nevertheless, the arguments described here are also involved in the formal security proof.

Theorem 1 (informal). *The protocol* $\mathtt{DAA_{OPT}}$ *is secure under the DBDH, DDH*$_{\mathbb{G}_1}$ *and q-SDH assumptions in the random oracle model.*

Proof (Sketch). We argue that $\mathtt{DAA_{OPT}}$ is anonymous, unforgeable and non-frameable as follows.

Anonymity. The SPK_2 constructed in the sign protocol of $\mathtt{DAA_{OPT}}$ is zero-knowledge by programming random oracles H_1 and H_2. Thus, there exists a simulator that can simulate a proof π_2 of SPK_2 for any statement, and no adversary can notice the difference. To prove that signatures are unlinkable, we pick a fresh key $gsk \xleftarrow{\$} \mathbb{Z}_p$ for $\mathsf{bsn} = \perp$ or a new basename $\mathsf{bsn} \neq \perp$, compute an unlinkable-tag/pseudonym (B, K) with gsk, and simulate a proof π_2 of SPK_2 in every signature generation of honest platforms. This is indistinguishable using a hybrid argument, where in the i-th game hop we use a fresh key gsk_i every time that the honest platform signs with $\mathsf{bsn}_i = \perp$ (or a new basename $\mathsf{bsn}_i \neq \perp$).

We prove that the i-th game hop is indistinguishable from the $(i-1)$-th one under the DDH$_{\mathbb{G}_1}$ assumption if $\mathsf{bsn}_i = \perp$ and the DBDH assumption otherwise. For the case of $\mathsf{bsn}_i = \perp$, given a DDH$_{\mathbb{G}_1}$ instance $(\bar{g}, \bar{g}^\alpha, \bar{g}^\beta, \bar{g}^\chi)$ with either $\chi = \alpha\beta$ or $\chi \xleftarrow{\$} \mathbb{Z}_p$, we simulate as follows. We set $\bar{g}^\alpha$ as the TPM's public key tpk and simulate a proof π_t due to the zero-knowledge property of SPK_t, and choose $hsk \xleftarrow{\$} \mathbb{Z}_p$ as the host's secret key. When signing with $\mathsf{bsn}_i = \perp$, we simulate a proof π_2 of SPK_2, and set $B = \bar{g}^\beta$ and $K = \bar{g}^\chi \cdot (\bar{g}^\beta)^{hsk}$. If $\chi = \alpha\beta$, the same key was used to sign, and if $\chi \xleftarrow{\$} \mathbb{Z}_p$, a fresh key was used. For the case that $\mathsf{bsn}_i \neq \perp$ and bsn_i is a new basename, given a DBDH instance $(g_1, g_2, g_1^\alpha, g_2^\beta, g_1^\delta, g_2^\delta, e(g_1, g_2)^\chi)$ with either $\chi = \alpha\beta\delta$ or $\chi \xleftarrow{\$} \mathbb{Z}_p$, we simulate as follows. We set g_1^δ as $\bar{g}$ and the unknown α as the key gsk of the platform. We can choose $tsk \xleftarrow{\$} \mathbb{Z}_p$ as the TPM's secret key, and pick $C \xleftarrow{\$} \mathbb{G}_1$ and simulate a proof π_h, as SPK_h is zero-knowledge. We also program the random oracle such that $H_{\mathbb{G}_2}(\mathsf{bsn}_i) = g_2^\beta$. When signing with bsn_i, we simulate a proof π_2 and set $K = e(g_1, g_2)^\chi$ as the pseudonym. If $\chi = \alpha\beta\delta$, the same key was used to sign, and if $\chi \xleftarrow{\$} \mathbb{Z}_p$, a fresh key was used.

Now, for any signature of honest platforms, an unlinkable-tag/pseudonym is computed using a fresh key, a proof π_2 is simulated. Besides, T_1 is uniformly random in $\mathbb{G}_1^*$, $T_2 = T_1^\gamma$ and Y' is uniformly random in $\mathbb{G}_1$, and thus they do not involve any information about the honest platform. Therefore, no adversary could break the anonymity of $\mathtt{DAA_{OPT}}$.

Unforgeability. First, we argue that no adversary could forge signatures using a credential cre from a platform with an honest TPM even if the host is corrupted. Signatures in our protocol $\mathtt{DAA_{OPT}}$ include the proofs of SPK_2 which prove knowledge of secret key $gsk = tsk + hsk$. Then, the adversary must know secret key tsk if it uses the credential cre, as SPK_2 is a proof of knowledge. This is infeasible under the Discrete-Logarithm (DL) assumption implied by the assumptions in Theorem 1, where the security analysis is very similar to the one in the non-frameability and omitted here. Second, a platform proves that $K = B^{gsk}$ is constructed correctly using the same key from its credential via SPK_2. If key gsk is added to the revocation list $\mathtt{RL}$, the private revocation check would reject all signatures created by gsk.

Next, we only need to show that no adversary could forge signatures using a credential that were not issued by the honest issuer. We can reduce this to existential unforgeability against adaptive chosen message attacks (EUF-CMA) of the BBS+ signature, which has been proved under the q-SDH assumption [ASM06,CDL16a].

Specifically, for the issuance of a credential, we can extract a platform secret key gsk and a randomness u' from proofs π_t and π_h of SPK_t and SPK_h, and then make a query gsk to the signing oracle and obtain a BBS+ signature (A, x, u). Then we can issue the corresponding credential $(A, x, , u - u')$ to the platform. When we extract a platform secret key and credential from a forged signature, the key was not signed by the issuer, then the key and credential must be a forgery of the BBS+ signature scheme.

Non-frameability. We argue that an honest platform cannot be framed under the DL assumption, even though the issuer is corrupted. Given a DL instance $(\bar{g}, \bar{g}^{\alpha})$, we set $\bar{g}^{\alpha}$ as the TPM's public key tpk and pick $hsk \xleftarrow{\$} \mathbb{Z}_p$ as the host's secret key. Then, we simulate a proof π_t of SPK_t by programming the random oracle H_1 in every execution of the join or sign protocol associated with the honest platform. If the adversary forges a signature which links to a signature of the honest platform, it must prove knowledge of the secret key gsk of the platform. We can extract the key gsk from the proof π_2 in the forged signature, and output $gsk - hsk$ as the discrete logarithm α which breaks the DL assumption. □

In the full formal security proof as described in Appendix C, we rewind to extract the witnesses from the proofs of SPK_1, SPK_t, SPK_h and SPK_2, which is in line with the security proofs of recent DAA schemes with Fiat-Shamir proofs in the UC model [CDL16a,CCD+17]. Camenisch et al. [CDL16b,CDL16a] also consider that instantiating the SPKs to be online extractable via combining Paillier encryption [Pai99] with Fiat-Shamir proofs [FS86]. However, the instantiation is considerably more expensive, and is not compatible with TPM 2.0. As in [CDL16a,CCD+17], we prove that $\mathtt{DAA_{OPT}}$ satisfies the stand-alone security instead of UC security when instantiating the underlying SPKs by Fiat-Shamir proofs and rewinding for extraction. As a result, we require that the join protocol is executed sequentially for the security proof.

5 TPM 2.0 Implementation of Our DAA Scheme

We show how to implement our DAA scheme $\mathtt{DAA_{OPT}}$ using the TPM commands specified in the TPM 2.0 specification [Tru16]. Specifically, we first give a brief description of the TPM 2.0 commands that will be used to implement $\mathtt{DAA_{OPT}}$, and refer the reader to TPM 2.0 [Tru16] for details. Then, we show how to implement $\mathtt{DAA_{OPT}}$ using these TPM 2.0 commands.

TPM 2.0 allows different types of signatures (e.g., ECDAA, ECSchnorr and U-Prove) to use the same TPM commands, which is achieved by splitting a signing procedure into two TPM commands: the first one is TPM2_Commit() that produces a commitment and the second one is a signing command. The signing command has several versions, dependent on what the signature is used for. As examples, we consider three DAA use cases as follows:

- Use Case I (corresponding to Application I): a DAA signature is used to quote PCR values, and a TPM 2.0 command TPM2_Quote() should be invoked.
- Use Case II (corresponding to Application II): a DAA signature is used to certify a key created by the TPM, and a TPM 2.0 command TPM2_Certify() should be invoked.
- Use Case III (corresponding to Application III): a DAA signature is used to sign an arbitrary message provided by the host, and a TPM 2.0 command TPM2_Sign() should be invoked.

In Use Cases I and II, a message m to be signed consists of two parts: a TPM message m_t (i.e., either PCR values or a TPM key) and a host message m_h (e.g., a nonce from a verifier). In Use Case III, a message m to be signed is totally provided by the host.

5.1 Outline of TPM 2.0 Commands

Following the TPM 2.0 specification [Tru16], cryptographic keys are stored in a key hierarchy, which includes a root key, an arbitrary number of layers of storage keys and one layer of leaf keys. Usually, only the root key is stored inside the TPM. Each other key has a parent key in one layer above this key, and each key protects at least one child key except for leaf keys. A leaf key is used for encryption/decryption, signing/verification or key exchange. A TPM makes use of a key with the following three items:

– **Key handle:** A key handle is a 32-bit random value uniquely identifying a key and connects multiple commands that use this key. A key handle is issued by the TPM when the key is loaded into the TPM. When the key is subsequently used in another command, the handle is taken as input for this command. If more than one key are involved in a command, all handles of these keys are taken as input for the command. After a key handle is released, the key needs to be re-loaded for the next use.

– **Key name:** The name of an asymmetric key is used for identifying the key externally, and it is a hash digest of the public portion of the key. We use $tpk.name$ to denote the key name of a public key tpk.

– **Key blob:** A key stored outside of the TPM is in a format of a key blob that is associated with its parent key PKEY. For an asymmetric key pair, written as $tk = (tpk, tsk)$ with the public and private portions, the key blob is

$$(tk)^* = \big((tsk)_{\text{SK}}, tpk, \mathsf{MAC}_{\text{MK}}((tsk)_{\text{SK}} \| tpk.name)\big),$$

where $(tsk)_{\text{SK}}$ is a symmetric-encryption ciphertext on plaintext tsk under the encrytion key SK, $\mathsf{MAC}_{\text{MK}}(\cdot)$ is a message authentication code (MAC) under a key MK, and (SK, MK) is derived from the parent key PKEY using a key derivation function, i.e., $(\text{SK}, \text{MK}) \leftarrow \mathsf{KDF}(\text{PKEY}, \text{SALT})$; SALT is used to make PKEY reusable. For simplicity, we will omit the salt from $\mathsf{KDF}(\text{PKEY}, \text{SALT})$ in the rest part of this section.

When tk is used as a TPM DAA signing key or any other TPM signing keys, it has a property named as *restricted* or *unrestricted*. A restricted signing key is used to quote PCR values, to certify a TPM key, or to sign a TPM computed hash digest. An unrestricted key can be used to sign any given message. Therefore, a message signed under an *unrestricted* key cannot be claimed that this is a set of PCR values, a key created by the TPM etc. A TPM key tk must be restricted for Use Cases I and II, and it is either restricted or unrestricted for Use Case III.

In the following description of TPM 2.0 commands, we continue using $\mathbb{G}_1 = \langle \bar{g} \rangle$ to denote a group of prime order p with a fixed generator $\bar{g}$. Let $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$ be a cryptographic hash function used by a TPM. To implement $\mathtt{DAA_{OPT}}$, we recommend using the following TPM 2.0 commands:

– Both TPM2_Create() and TPM2_CreatePrimary() are used to create a TPM key $tk = (tpk, tsk)$. On input TPM2_Create() : the TPM does the following:
  1. Choose a fresh secret key $tsk \xleftarrow{\$} \mathbb{Z}_p$ and compute a public key $tpk \leftarrow \bar{g}^{tsk}$.
  2. Set a *restricted* or *unrestricted* attribute for the key.
  3. Generate and output a key blob $(tk)^*$.

  The difference of using TPM2_CreatePrimary() is that instead of creating a key from a random number, TPM2_CreatePrimary() creates a key from a TPM secret seed using a KDF. To simplify the writing, we will use TPM2_Create() only in the remaining part of this paper.

– TPM2_Load() is used to load a key into the TPM. On input TPM2_Load($(tk)^*$) : the TPM takes as input a key blob $(tk)^*$ and its parent key handle, from that the TPM finds the parent key PKEY, which has already been loaded into the TPM. The TPM then generates $(\text{SK}, \text{MK}) \leftarrow \mathsf{KDF}(\text{PKEY})$, computes $tpk.name$ from tpk and checks the validity of $\mathsf{MAC}_{\text{MK}}((tsk)_{\text{SK}} \| tpk.name)$. The TPM decrypts $(tsk)_{\text{SK}}$, and checks if (tpk, tsk) forms a valid key pair. If the checks pass, the TPM outputs a key handle $tk.handle$ along with the key name $tpk.name$. Now tk is stored inside the TPM and can be used for future operations.

– Several TPM 2.0 hash commands allow a TPM to compute a hash digest with different message lengths. If the message is not longer than one hash block, use TPM2_Hash(). Otherwise, use a set of commands to handle sequences. In this paper, we use TPM2_Hash() only to implement $\mathtt{DAA_{OPT}}$. On input TPM2_Hash(msg) : with a message msg given by the host, the TPM does the following:
  1. Check that the first octets of message msg are not "TPM_GENERATED_VALUE".
  2. If the check passes, compute a digest $c_t \leftarrow \mathsf{H}(msg)$ and a "TPMT_TK_HASHCHECK" ticket τ which is a MAC on message c_t.
  3. Output (c_t, τ).

  The TPM also has an internal hash operation that can handle a message m_t generated by the TPM, such as PCR values or a TPM key. In this case, the message will start with the label "TPM_GENERATED_VALUE".

– TPM2_Commit() is the first TPM command in the TPM signing procedure. On input TPM2_Commit($P_1, \mathsf{s}_2, \mathsf{y}$) : the TPM executes as follows:

1. If $P_1 \neq \perp$, check whether $P_1 \in \mathbb{G}_1$ or not.
2. If $(\mathsf{s}_2, \mathsf{y}) \neq \perp$, compute $\mathsf{x} = H(\mathsf{s}_2)$ for a cryptography hash function H, and then set $B \leftarrow (\mathsf{x}, \mathsf{y})$ and check whether $B \in \mathbb{G}_1$ or not. The string s_2 may contain a basename bsn for DAA.
3. If the above checks fail, output an error and abort.
4. Set $E, K, L \leftarrow \perp$.
5. Pick $r \xleftarrow{\$} \mathbb{Z}_p$ and store (ctr, r) in a list `Committed`, where ctr is a counter used to retrieve r. Here, we assume that `Committed` and ctr are initialized as $\emptyset$ and 0 respectively.
6. If $P_1 \neq \perp$, compute $E \leftarrow P_1^r$.
7. If $(\mathsf{s}_2, \mathsf{y}) \neq \perp$, compute $K \leftarrow B^{tsk}$ and $L \leftarrow B^r$.
8. If $P_1 = \perp$ and $(\mathsf{s}_2, \mathsf{y}) = \perp$, compute $E \leftarrow \bar{g}^r$.
9. Increment ctr and output (E, K, L, ctr).

The second TPM command in the TPM signing procedure, as we discussed before, has three cases: $\mathsf{TPM2\_Sign}()$, $\mathsf{TPM2\_Certify}()$ and $\mathsf{TPM2\_Quote}()$, dependent on what the signature is used for.

– On input $\mathsf{TPM2\_Sign}(c_t, \tau, ctr)$: the TPM executes as follows:
  1. If the TPM key is *unrestricted* and $\tau = \perp$, check that the size of c_t is equal to the output length of H. Otherwise, check the validity of ticket τ.
  2. If the above check passes, execute the following $\mathsf{CryptSign}(c_t, ctr)$ function:[5]
     (a) Retrieve a pair (ctr, r) and remove it from list `Committed`, output an error if no such pair was found.
     (b) Pick $N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$ and compute $c \leftarrow \mathsf{H}(N_t, c_t)$.
     (c) Compute $s \leftarrow r + c \cdot tsk \mod p$ and output (N_t, s).
– On input $\mathsf{TPM2\_Certify}(qualifyData, keyhandle, ctr)$: Given an extra data *qualifyData*, a *keyhandle* and a counter ctr, the TPM retrieves a public key m_t using the key handle *keyhandle*, and does the following:
  1. Compute a hash digest $c_t \leftarrow \mathsf{H}(qualifyData, \mathsf{H}(\text{“TPM\_GENERATED\_VALUE”}, m_t))$.
  2. Execute the $\mathsf{CryptSign}(c_t, ctr)$ function as described in $\mathsf{TPM2\_Sign}()$ to obtain (N_t, s).
  3. Output (N_t, s).
– On input $\mathsf{TPM2\_Quote}(qualifyData, PCRselect, ctr)$: the TPM executes as follows:
  1. Select the corresponding PCR values m_t from the platform configuration register according to *PCRselect*, and compute a hash digest of m_t denoted by *pcrDigest*.
  2. Compute a hash digest $c_t \leftarrow \mathsf{H}(qualifyData, \mathsf{H}(\text{“TPM\_GENERATED\_VALUE”}, pcrDigest))$.
  3. Execute the $\mathsf{CryptSign}(c_t, ctr)$ function as described in $\mathsf{TPM2\_Sign}()$ to obtain (N_t, s).
  4. Output (N_t, s) and *pcrDigest*.
– $\mathsf{TPM2\_ActivateCredential}()$ is used to allow the DAA issuer to authenticate the public key *tpk* of a TPM and to issue a credential cre' and a number of attributes *attrs* confidentially in the join protocol by using the endorsement key $ek = (epk, esk)$ of the TPM. Given an endorsement public key *epk* and a TPM public key *tpk*, the issuer generates a fresh secret seed *seed* and a fresh symmetric encryption key k, and then computes an encryption blob $(ct)^*$ as follows:

$$(ct)^* = \mathsf{ENC}_{epk}(tpk, k) = \big((seed)_{epk}, (k)_{\text{SK}}, \mathsf{MAC}_{\text{MK}}((k)_{\text{SK}} \| tpk.name)\big)$$

  where $(seed)_{epk}$ is a public-encryption ciphertext on message *seed* under public key *epk*, $(\text{SK}, \text{MK}) \leftarrow \mathsf{KDF}(seed)$ and $(k)_{\text{SK}}$ is a symmetric-encryption ciphertext on message k under secret key SK. Additionally, the issuer generates a symmetric-encryption ciphertext $(cre' \| attrs)_k$ on message $cre' \| attrs$ under key k. On input $\mathsf{TPM2\_ActivateCredential}(ek.handle, tk.handle, (ct)^*)$: the TPM executes as follows:
  1. Retrieve a secret key *esk* using a key handle *ek.handle*, and decrypt $(seed)_{epk}$ with *esk* to obtain *seed*.
  2. Derive a symmetric key SK and a MAC key MK, i.e, $(\text{SK}, \text{MK}) \leftarrow \mathsf{KDF}(seed)$.
  3. Retrieve a key name *tpk.name* using a key handle *tk.handle* and compute $\mathsf{MAC}_{\text{MK}}((k)_{\text{SK}} \| tpk.name)$.
  4. Check whether the computed MAC value matches the one in encryption blob $(ct)^*$.
  5. If the check fails, output an error. Otherwise, decrypt $(k)_{\text{SK}}$ with SK and output k.

  When the TPM releases k, the host can decrypt $(cre' \| attrs)_k$ with key k to obtain a credential cre' and its attributes *attrs* from the issuer.

In the above description of $\mathsf{TPM2\_Commit}()$, $\mathsf{TPM2\_Sign}()$, $\mathsf{TPM2\_Certify}()$ and $\mathsf{TPM2\_Quote}()$, we assume an expensive implementation, i.e., the TPM maintains internally a list `Committed`, and a counter ctr and a randomness r produced in $\mathsf{TPM2\_Commit}()$ are stored in `Committed`. The TPM 2.0 specification [Tru16] provides an alternative efficient implementation without storing the list, which is suggested by David Wooten and described in [CL13]. We refer the reader to [Tru16,CL13] for the concrete implementation.

[5] Note that a nonce N_t has been added to the CryptSign function in the revision 01.38 of TPM 2.0 specification [Tru16].

5.2 The TPM 2.0 Implementation of Our DAA Scheme

For the sake of simplicity, we consider that a key $tk = (tpk, tsk)$ is always loaded into the TPM via $\mathsf{TPM2\_Load}()$, before it would be used. Thus, we could omit the invocation of $\mathsf{TPM2\_Load}()$ in the description of implementing our scheme $\mathtt{DAA_{OPT}}$.

Below, we present how to use the TPM 2.0 commands described in Section 5.1 to implement the `TPM.Create`, `TPM.Commit` and `TPM.Sign` procedures in $\mathtt{DAA_{OPT}}$.

- For the `TPM.Create` procedure, the host $\mathcal{H}_j$ calls a TPM command $\mathsf{TPM2\_Create}()$, and the TPM $\mathcal{M}_i$ outputs a key blob $(tk)^*$ including a public key tpk.
- For the `TPM.Commit` procedure, $\mathcal{H}_j$ calls a TPM command $\mathsf{TPM2\_Commit}(\perp, \perp)$, and $\mathcal{M}_i$ outputs a commitment $E = \bar{g}^r$ and a counter ctr.
- For the `TPM.Sign` procedure in the *join* protocol, we consider two cases relying on whether a signing key tsk is restricted or not.
  1. If the TPM secret key tsk is *restricted*, host $\mathcal{H}_j$ calls a TPM command $\mathsf{TPM2\_Hash}(c_h)$, and TPM $\mathcal{M}_i$ outputs a digest c_t and a ticket τ. Then, $\mathcal{H}_j$ calls $\mathsf{TPM2\_Sign}(c_t, \tau, ctr)$, and $\mathcal{M}_i$ outputs (N_t, s).
  2. If the TPM secret key tsk is *unrestricted*, host $\mathcal{H}_j$ calls $\mathsf{TPM2\_Sign}(c_h, \perp, ctr)$, and TPM $\mathcal{M}_i$ outputs (N_t, s).
- For the `TPM.Sign` procedure in the *sign* protocol, we consider three DAA use cases as follows.
  1. For Use Case I, host $\mathcal{H}_j$ first computes a hash digest $d_h \leftarrow \mathsf{H}(\text{"qualifyingData"}, m_h, \mathsf{bsn}, D, I, c_h)$, and then calls $\mathsf{TPM2\_Quote}(d_h, PCRselect, ctr)$. TPM $\mathcal{M}_i$ outputs (N_t, s) along with $pcrDigest$.
  2. For Use Case II, the host $\mathcal{H}_j$ loads the key to be certified into the TPM by calling a TPM command $\mathsf{TPM2\_Load}()$ to receive a key handle $keyhandle$. Then, $\mathcal{H}_j$ computes a hash digest $d_h \leftarrow \mathsf{H}(\text{"qualifyingData"}, m_h, \mathsf{bsn}, D, I, c_h)$ and calls a TPM command $\mathsf{TPM2\_Certify}(d_h, keyhandle, ctr)$. The TPM $\mathcal{M}_i$ outputs (N_t, s).
  3. For Use Case III, we distinguish which type the TPM secret key tsk belongs to.
     (a) If tsk is *restricted*, host $\mathcal{H}_j$ computes $d_h \leftarrow \mathsf{H}(\text{"hostMessage"}, m, \mathsf{bsn}, D, I, c_h)$, and then calls a TPM command $\mathsf{TPM2\_Hash}(d_h)$. TPM $\mathcal{M}_i$ outputs a digest c_t and a ticket τ. Then, $\mathcal{H}_j$ calls a TPM command $\mathsf{TPM2\_Sign}(c_t, \tau, ctr)$ and $\mathcal{M}_i$ outputs (N_t, s).
     (b) If tsk is *unrestricted*, host $\mathcal{H}_j$ can compute $c_t \leftarrow \mathsf{H}(m, \mathsf{bsn}, D, I, c_h)$ by itself. Then, $\mathcal{H}_j$ calls a TPM command $\mathsf{TPM2\_Sign}(c_t, \perp, ctr)$ and TPM $\mathcal{M}_i$ outputs (N_t, s).

  In the above TPM 2.0 implementation, we let the host compute the hash digest of m_h (or m) and bsn, D, I, c_h to achieve better performance. This has no impact for the security even if the host is corrupted, since the simulator controls the random oracle H, can extract a tuple $(m_h/m, \mathsf{bsn}, D, I, c_h)$ from the H-list maintained by itself, and send the tuple to the ideal functionality in the security proof. Depending on the use case and the type of the protocol, the hash function H_1 used by the TPM in the construction of our scheme $\mathtt{DAA_{OPT}}$ has different ways of implementation, which would be explicit from the application scenario and that either the join protocol or the sign protocol is executed by a platform.

Now, we show how to use a TPM command $\mathsf{TPM2\_ActivateCredential}()$ to establish the semi-authenticated channel between the TPM and issuer in the join protocol, by following the description in [CL13].

1. A host $\mathcal{H}_j$ sends an endorsement public key epk and a public key tpk to an issuer $\mathcal{I}$ as the `JOIN` request.
2. Upon receiving epk and tpk, $\mathcal{I}$ checks the validity of epk via validating the certificate of epk. If the check passes, $\mathcal{I}$ picks a nonce $N_I \leftarrow \{0,1\}^\lambda$ and generates an encryption blob $(ct_1)^* \leftarrow \mathsf{ENC}_{epk}(tpk, N_I)$. Then $\mathcal{I}$ sends $(ct_1)^*$ to $\mathcal{H}_j$.
3. $\mathcal{H}_j$ calls $\mathsf{TPM2\_ActivateCredential}(ek.handle, tk.handle, (ct_1)^*)$ and the TPM $\mathcal{M}_i$ outputs N_I, where endorsement key ek and TPM key tk are assumed to have been loaded into $\mathcal{M}_i$ via $\mathsf{TPM2\_Load}()$.
4. Upon receiving a tuple (C, π_t, π_h) and a nonce N_I, $\mathcal{I}$ checks the validity of N_I and proofs π_t, π_h. If the check passes, $\mathcal{I}$ generates a credential $cre' \leftarrow (A, x, u'')$ and a number of attributes $attrs = (a_1, \ldots, a_n)$. Then $\mathcal{I}$ creates a fresh key k, and generates an encryption blob $(ct_2)^* \leftarrow \mathsf{ENC}_{epk}(tpk, k)$ and a symmetric-encryption ciphertext $sc \leftarrow (cre' \| attrs)_k$. $\mathcal{I}$ sends $((ct_2)^*, sc)$ to $\mathcal{H}_j$.
5. $\mathcal{H}_j$ calls $\mathsf{TPM2\_ActivateCredential}(ek.handle, tk.handle, (ct_2)^*)$ and the TPM $\mathcal{M}_i$ outputs k. $\mathcal{H}_j$ decrypts ciphertext sc with key k and obtains $cre' = (A, x, u'')$ and $attrs = (a_1, \ldots, a_n)$.

6 Performance Evaluation

In this section, we compare the efficiency of our scheme $\mathtt{DAA_{OPT}}$ with the existing DAA schemes supported by the TPM 2.0 specification [Tru16]. We use CPS, BL and CDL to denote these DAA schemes, where CPS is based on the LRSW-DAA scheme [CPS10], BL is based on the SDH-DAA scheme [BL10b], and CDL is the SDH-DAA scheme in the full version of [CDL16a] but removes the session identifiers for UC security. In particular, we evaluate the efficiency of BL when considering the efficiency improvement of this scheme using this optimization in [CU15]. We also compare the efficiency of these DAA schemes with the functionality extension of attributes, where CPS and BL can be extended to support attributes following [CU15], and CDL provides the support of attributes by itself. For fairness, we consider that all the DAA schemes let the host store all attributes and the TPM protect the secret key only. We refer the reader to [CL13,CU15] for the implementation details of CPS and BL using the TPM 2.0 commands. In all our comparisons, we can directly obtain the efficiency of traditional DAA schemes when setting both the number of attributes n and the number of undisclosed attributes u as zero.

Table 1. Efficiency Comparison of the Sign Protocol and Verification Algorithm*

<table>
<tr><th rowspan="3" colspan="2">DAA Scheme†</th><th colspan="4">Sign Protocol</th><th rowspan="3">Verification‡</th></tr>
<tr><th colspan="2">TPM</th><th colspan="2">Host</th></tr>
<tr><th>Total</th><th>Online</th><th>Total</th><th>Online</th></tr>
<tr><td rowspan="2">CPS</td><td>$\mathsf{bsn} = \perp$</td><td>$1E_{\mathbb{G}_1}$</td><td>$\mathtt{H} + \mathtt{mul}$</td><td>$4E_{\mathbb{G}_1} + nE_{\mathbb{G}_1} + 1E^u_{\mathbb{G}_1}$</td><td>–</td><td>$1E^{2+n}_{\mathbb{G}_1} + 1E^n_{\mathbb{G}_{1(t)}} + 1E^n_{\mathbb{G}_{2(t)}} + 4P + [2P]$</td></tr>
<tr><td>$\mathsf{bsn} \neq \perp$</td><td>$3E_{\mathbb{G}_1}$</td><td>$3E_{\mathbb{G}_1}$</td><td>$4E_{\mathbb{G}_1} + nE_{\mathbb{G}_1} + 1E^u_{\mathbb{G}_1}$</td><td>–</td><td>$1E^2_{\mathbb{G}_1} + 1E^{2+n}_{\mathbb{G}_1} + 1E^n_{\mathbb{G}_{1(t)}} + 1E^n_{\mathbb{G}_{2(t)}} + 4P + [2P]$</td></tr>
<tr><td rowspan="2">BL</td><td>$\mathsf{bsn} = \perp$</td><td>$3E_{\mathbb{G}_1}$</td><td>$\mathtt{H} + \mathtt{mul}$</td><td>$2E_{\mathbb{G}_1} + 1E^{2+u}_{\mathbb{G}_1} + 2P$</td><td>–</td><td>$1E^2_{\mathbb{G}_1} + 1E^{4+n}_{\mathbb{G}_1} + 2P$</td></tr>
<tr><td>$\mathsf{bsn} \neq \perp$</td><td>$3E_{\mathbb{G}_1}$</td><td>$3E_{\mathbb{G}_1}$</td><td>$2E_{\mathbb{G}_1} + 1E^{2+u}_{\mathbb{G}_1} + 2P$</td><td>$1P$</td><td>$1E^2_{\mathbb{G}_1} + 1E^{4+n}_{\mathbb{G}_1} + 2P$</td></tr>
<tr><td rowspan="2">CDL</td><td>$\mathsf{bsn} = \perp$</td><td>$3E_{\mathbb{G}_1}$</td><td>$\mathtt{H} + \mathtt{mul}$</td><td>$1E_{\mathbb{G}_1} + 3E^2_{\mathbb{G}_1} + 1E^{2+u}_{\mathbb{G}_1}$</td><td>–</td><td>$1E^2_{\mathbb{G}_1} + 1E^3_{\mathbb{G}_1} + 1E^{4+n}_{\mathbb{G}_1} + 2P$</td></tr>
<tr><td>$\mathsf{bsn} \neq \perp$</td><td>$3E_{\mathbb{G}_1}$</td><td>$3E_{\mathbb{G}_1}$</td><td>$1E_{\mathbb{G}_1} + 3E^2_{\mathbb{G}_1} + 1E^{2+u}_{\mathbb{G}_1}$</td><td>–</td><td>$1E^2_{\mathbb{G}_1} + 1E^3_{\mathbb{G}_1} + 1E^{4+n}_{\mathbb{G}_1} + 2P$</td></tr>
<tr><td rowspan="2">$\mathtt{DAA_{OPT}}$</td><td>$\mathsf{bsn} = \perp$</td><td>$1E_{\mathbb{G}_1}$</td><td>$\mathtt{H} + \mathtt{mul}$</td><td>$5E_{\mathbb{G}_1} + 3E^2_{\mathbb{G}_1} + 1E^{2+u}_{\mathbb{G}_1}$</td><td>–</td><td>$1E^2_{\mathbb{G}_1} + 1E^3_{\mathbb{G}_1} + 1E^{4+n}_{\mathbb{G}_1} + 2P$</td></tr>
<tr><td>$\mathsf{bsn} \neq \perp$</td><td>$1E_{\mathbb{G}_1}$</td><td>$\mathtt{H} + \mathtt{mul}$</td><td>$2E_{\mathbb{G}_1} + 3E^2_{\mathbb{G}_1} + 1E^{2+u}_{\mathbb{G}_1} + 2P$</td><td>$2P$</td><td>$1E^3_{\mathbb{G}_1} + 1E^{4+n}_{\mathbb{G}_1} + 1E^2_{\mathbb{G}_T} + 2P$</td></tr>
</table>

* $E^m_{\mathbb{G}_i}$ $(i \in \{1, 2, T\})$: the cost of the product of m powers in $\mathbb{G}_i$; $E_{\mathbb{G}_i}$: the cost of one exponentiation in $\mathbb{G}_i$; P: the cost of a bilinear pairing. $E^m_{\mathbb{G}_{i(t)}}$ $(i \in \{1, 2\})$: the cost of one m-multi-exponentiations in group $\mathbb{G}_i$ with the size of the exponents being t such as a half of the size of p.

† The rows for $\mathsf{bsn} = \perp$ (resp., $\mathsf{bsn} \neq \perp$) represent the signing and verification costs about a fully anonymous (resp., pseudonymous) signature.

‡ $[X]$ denotes the *incremental* computational cost X when considering the support of attributes.

We first give a theoretical comparison by counting the number of costly operations and group elements in each DAA scheme, since the costly operations dominate the performance of DAA schemes. We implemented and benchmarked the TPM 2.0 commands needed for three DAA use cases on an Infineon TPM 2.0 chip. Then, we use the benchmark results along with the benckmark results for exponentiations and pairings over the BN curve in [BCN14] to evaluate the performance of $\mathtt{DAA_{OPT}}$ and other compared DAA schemes. We also give the comparison of concrete sizes of credentials and signatures over two kinds of BN curves recommended by the TCG. We omit the efficiency comparison of the join protocol, since the join protocol is executed much less frequently than the sign protocol or the verification algorithm.

6.1 Theoretical Comparison

We compare the efficiency of the signing protocol and verification algorithm of the DAA schemes supported by TPM 2.0 in Table 1, where the online signing cost for the host is obtained by assuming that attribute disclosure (D, I) is allowed to be selected offline. We count the computational costs of a hash function and a modular multiplication $r + c \cdot tsk \mod p$ for the TPM (denoted by H and mul) in Table 1, since they are still expensive for the TPM. In contrast, these computational costs are ignored for the host signing and verification algorithm, as they are very fast and much more efficient than exponentiations for the host and verifier with much more powerful computational capability.

From Table 1, we can see that our scheme $\mathtt{DAA_{OPT}}$ is the only scheme achieving the fully optimal TPM signing efficiency. The online signing cost of the TPM for pseudonymous signature mode in $\mathtt{DAA_{OPT}}$ is much less than other DAA schemes. The verification cost in Table 1 does not include private key revocation. In terms of the efficiency of private key revocation, $\mathtt{DAA_{OPT}}$ has the same efficiency as other three DAA schemes for fully anonymous signatures, and provides the same on-line efficiency as other three schemes for pseudonymous signatures, since the verifier can pre-compute $e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^{gsk}$ for each $gsk \in \mathtt{RL}$.

Table 2. Theoretical Comparison of the Sizes of Credentials and Signatures*

DAA Scheme	Credential Size	Signature Size		Security Proof
		$\mathsf{bsn} = \perp$	$\mathsf{bsn} \neq \perp$	
CPS	$4\lvert\mathbb{G}_1\rvert + n\lvert\mathbb{G}_1\rvert$	$4\lvert\mathbb{G}_1\rvert + 3\lvert\mathbb{Z}_p\rvert + n\lvert\mathbb{G}_1\rvert + u\lvert\mathbb{Z}_p\rvert$	$5\lvert\mathbb{G}_1\rvert + 3\lvert\mathbb{Z}_p\rvert + n\lvert\mathbb{G}_1\rvert + u\lvert\mathbb{Z}_p\rvert$	no
BL	$1\lvert\mathbb{G}_1\rvert + 1\lvert\mathbb{Z}_p\rvert$	$3\lvert\mathbb{G}_1\rvert + 6\lvert\mathbb{Z}_p\rvert + u\lvert\mathbb{Z}_p\rvert$	$2\lvert\mathbb{G}_1\rvert + 6\lvert\mathbb{Z}_p\rvert + u\lvert\mathbb{Z}_p\rvert$	no
CDL	$2\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{Z}_p\rvert$	$5\lvert\mathbb{G}_1\rvert + 7\lvert\mathbb{Z}_p\rvert + u\lvert\mathbb{Z}_p\rvert$	$4\lvert\mathbb{G}_1\rvert + 7\lvert\mathbb{Z}_p\rvert + u\lvert\mathbb{Z}_p\rvert$	yes
$\mathtt{DAA_{OPT}}$	$3\lvert\mathbb{G}_1\rvert + 3\lvert\mathbb{Z}_p\rvert$	$5\lvert\mathbb{G}_1\rvert + 7\lvert\mathbb{Z}_p\rvert + u\lvert\mathbb{Z}_p\rvert$	$3\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + 7\lvert\mathbb{Z}_p\rvert + u\lvert\mathbb{Z}_p\rvert$	yes

* $\lvert\mathbb{G}\rvert$: the bit-length of an element in group $\mathbb{G}$.

In Table 2, we compare the sizes of credentials and signatures, where the bit-length ℓ_n of a nonce N_t is counted as $\lvert\mathbb{Z}_p\rvert$. While the SDH-DAA schemes $\mathtt{DAA_{OPT}}$, BL and CDL have $\mathcal{O}(1)$ credential size, the LRSW-DAA scheme CPS has $\mathcal{O}(n)$ credential size. Moreover, when supporting attributes, the *incremental* size of signatures in the SDH-DAA schemes is much less than CPS. We also compare the security of these DAA schemes in Table 2. While CDL and $\mathtt{DAA_{OPT}}$ are provably secure in the UC security model, no security proof is found for CPS and BL with/without attributes in a valid security model [BFG+13,CU15,CDL16b,CDL16a].

6.2 Efficiency Comparison Using Benchmark Results

We present the benchmark results for the following TPM 2.0 commands: TPM2_Commit(), TPM2_Quote(), TPM2_Certify(), TPM2_Sign() and TPM2_Hash(), by implementing them on an Infineon TPM 2.0 chip with vendor ID IFXSLB9670. The TPM 2.0 chip is installed on a module designed for the Raspberry Pi. The program used to obtain the timings was running on a Raspberry Pi 3 fitted with the Infineon TPM module, and was compiled using g++ 6.3.0. The Raspberry Pi 3 is equipped with a 64-bit ARMv7 processor, but the operating system Raspbian (version 4.14.30) runs in 32-bit mode. We adopt SHA256 to implement the hash function H used by the TPM chip.

The TCG recommended two types of Barreto-Naehrig (BN) curves [BN06] (i.e., BN_P256 and BN_P638) to support bilinear pairings. These BN elliptic curves have the form $y^2 = x^3 + b$ with embedding degree 12, where $b = 3$ for BN_P256 and $b = 257$ for BN_P638. According to the state-of-art analysis results [KB16,BD18], the BN_P256 curve only achieves about 100-bit security level, and the BN_P638 curve will provide more than 128-bit security level. Currently, only the BN_P256 curve is implemented on the TPM 2.0 chips, and the implementation of the BN_P638 curve has *not* been available for TPM 2.0 chips. Therefore, we only consider the BN_P256 curve for the comparison of computational efficiency. However, we will adopt both the BN_P256 and BN_P638 curves for the comparison of the sizes of credentials and signatures.

We consider five cases for the implementation of the TPM2_Commit() command:

Case 1. No input, i.e., $P_1 = \perp$ and $(\mathsf{s}_2, \mathsf{y}) = \perp$. Our scheme $\mathtt{DAA_{OPT}}$ uses TPM2_Commit() in this case.

Case 2. A single elliptic curve point $P_1 \neq \perp$ is input, but $(\mathsf{s}_2, \mathsf{y}) = \perp$. The LRSW-DAA scheme CPS uses this case with a random P_1 to generate fully anonymous signatures.

Case 3. Both a curve point $P_1 \neq \perp$ and $(\mathsf{s}_2, \mathsf{y}) \neq \perp$ are input, and P_1 is a random point. The LRSW-DAA scheme CPS uses this case to produce pseudonymous signatures.

Case 4. Only $(\mathsf{s}_2, \mathsf{y}) \neq \perp$ is input and $P_1 = \perp$. In this case, only $K = B^{tsk}, L = B^r$ are output. As far as we know, no DAA scheme uses this case.

Case 5. Both a curve point $P_1 \neq \perp$ and $(\mathsf{s}_2, \mathsf{y}) \neq \perp$ are input, and P_1 is a fixed base point. BL and CDL use this case to generate signatures for both $\mathsf{bsn} = \perp$ and $\mathsf{bsn} \neq \perp$.

Our benchmark results for the TPM 2.0 commands are shown in Table 3. The results are in milliseconds (ms) and averaged over 150 random instances. For TPM2_Quote(), only one PCR value is selected. The running time of TPM2_Certify() does not include the time creating a public key to be signed, where the public key is assumed to be created offline. These running times in Table 3 do not involve the time of communication between the TPM and host. Note that the communication overhead of the sign protocol in our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ is less than other DAA schemes.

Table 3. The Average Running Times of Several TPM 2.0 commands

TPM2_Commit()†				
Case 1	Case 2	Case 3	Case 4	Case 5
87.4 (0.2)	87.6 (0.6)	217.1 (0.5)	152.3 (0.8)	217.0 (0.5)
TPM2_Quote()			TPM2_Certify()	
50.2 (1.25)			50.1 (0.6)	
TPM2_Sign()			TPM2_Hash()	
49.8 (1.1)			23.0 (0.8)	

† The values in round brackets () is standard deviation.

We adopt the benchmark results by Bos et al. [BCN14] for the BN_P256 curve and an optimal ate pairing to estimate the performance of the host signing and the verification algorithm. Their benchmark results are obtained on a labtop with 2.9GHz Intel Core i7-3520M CPU averaged over thousands of random instances. Their benchmark results show that $1E_{\mathbb{G}_1}$, $1E_{\mathbb{G}_2}$, $1E_{\mathbb{G}_T}$ and $1P$ require about 0.3ms, 0.6ms, 1.1ms and 2.4ms respectively. The benchmark results [BCN14] are originally given in millions of clock cycles. We convert their results into milliseconds (ms) to match the measure of the running times of the TPM. We also show the sizes of elements (in bits) in groups $\mathbb{Z}_p$, $\mathbb{G}_1$ and $\mathbb{G}_T$ as follows: $|\mathbb{Z}_p| = 256$, $|\mathbb{G}_1| = 257$ and $|\mathbb{G}_T| = 3072$ over the BN_P256 curve; and $|\mathbb{Z}_p| = 638$, $|\mathbb{G}_1| = 639$ and $|\mathbb{G}_T| = 7656$ over the BN_P638 curve, when considering the point compression technique. Recently, Camenisch et al. [CDL17] also used benchmark results to estimate

Table 4. Comparison of Concrete Performance of the Sign Protocol and Verification Algorithm

DAA Scheme		Running Time of Sign Protocol (ms)								Verify (ms)
		TPM Signing		Host Signing			Total Signing			
		Total	Online	Total	Opt.*	Online	Total	Opt.*	Online	
CPS	$\mathsf{bsn} = \perp$	137.4	49.8	$1.2+$ $0.3n + 0.3u$	0.3	–	$138.6+$ $0.3n + 0.3u$	137.7	49.8	$10.2 + 0.75n+$ [4.8]
	$\mathsf{bsn} \neq \perp$	266.9	266.9	$1.2+$ $0.3n + 0.3u$	0.3	–	$268.1+$ $0.3n + 0.3u$	267.2	266.9	$10.8 + 0.75n+$ [4.8]
BL	$\mathsf{bsn} = \perp$	266.8	49.8	$6 + 0.3u$	2.4	–	$272.8 + 0.3u$	269.2	49.8	$6.6 + 0.3n$
	$\mathsf{bsn} \neq \perp$	266.8	266.8	$6 + 0.3u$	2.4	2.4	$272.8 + 0.3u$	269.2	269.2	$6.6 + 0.3n$
CDL	$\mathsf{bsn} = \perp$	266.8	49.8	$2.7 + 0.3u$	–	–	$269.5 + 0.3u$	266.8	49.8	$7.5 + 0.3n$
	$\mathsf{bsn} \neq \perp$	266.8	266.8	$2.7 + 0.3u$	–	–	$269.5 + 0.3u$	266.8	266.8	$7.5 + 0.3n$
$\mathtt{DAA}_{\mathtt{OPT}}$	$\mathsf{bsn} = \perp$	137.2	49.8	$3.9 + 0.3u$	0.3	–	$141.1 + 0.3u$	137.5	49.8	$7.5 + 0.3n$
	$\mathsf{bsn} \neq \perp$	137.2	49.8	$7.8 + 0.3u$	2.4	4.8	$145 + 0.3u$	139.6	54.6	$9.1 + 0.3n$

* The columns of "Opt." denote the signing times taking at the host or platform side, when the trick of parallel computation is involved and the number u of undisclosed attributes is not very large.

the efficiency of the host signing and the verification algorithm. This is acceptable as the fast operations (e.g., hash function and modular multiplication) have little impact on the running times over the host and verifier platforms with powerful computational capabilities. Using the benchmark results to estimate the performance of a scheme has also appeared in [AKS12,CDD17]. We do not consider the optimizations of multi-exponentiations and batch pairings, where they can be applied to all the DAA schemes and further reduce the times of the host signing and the verification algorithm.

In Table 4, we compare the performance of the sign protocol and verification algorithm using our benchmark results on an Infineon TPM 2.0 chip and the benchmark results on PCs [BCN14]. We only use the running time of TPM2_Sign() to evaluate the signing performance of the TPM for the sake of simplicity and clarity, since the three commands TPM2_Quote(), TPM2_Certify() and TPM2_Sign() have the almost same running times as seen in Table 3. We set the TPM key as *unrestricted*, and thus need not to invoke the TPM2_Hash() command when signing a message in Use Case III. This will obtain better signing efficiency, as we have seen that TPM2_Hash() takes about 23ms from Table 3. The columns of "Opt." in Table 4 represent not only the signing performance of DAA schemes with appropriate number of undisclosed attributes but also the signing efficiency of traditional DAA schemes.

Table 5. Comparison of the Sizes of Credentials and Signatures*

DAA Scheme	Credential Size (Bytes)		Signature Size (Bytes)			
			bsn $= \perp$		bsn $\neq \perp$	
CPS	$129+33n$	$320+80n$	$225+33n+32u$	$559+80n+80u$	$257+33n+32u$	$639+80n+80u$
BL	65	160	$289+32u$	$719+80u$	$257+32u$	$639+80u$
CDL	129	320	$385+32u$	$958+80u$	$353+32u$	$878+80u$
$\mathtt{DAA}_{\mathtt{OPT}}$	193	479	$385+32u$	$958+80u$	$705+32u$	$1755+80u$

* In each section, the left column denotes the size over the BN_P256 curve and the right column represents the size over the BN_P638 curve.

When the implementation trick of parallel computation is involved, our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ is about a factor 1.9x more efficient than other three DAA schemes for pseudonymous signature mode, and is about 1.9x faster than the SDH-DAA schemes BL and CDL for fully anonymous signature mode. In terms of online signing efficiency for pseudonymous signature mode, $\mathtt{DAA}_{\mathtt{OPT}}$ is about a factor 4.9x faster than other compared DAA schemes. Although CPS has the same signing efficiency as $\mathtt{DAA}_{\mathtt{OPT}}$ for fully anonymous signature mode, CPS is less efficient than $\mathtt{DAA}_{\mathtt{OPT}}$ in terms of the verification efficiency. For the verification efficiency, $\mathtt{DAA}_{\mathtt{OPT}}$ is the same as CDL and close to BL for fully anonymous signature mode, and 2.5ms/1.6ms slower than BL/CPS for pseudonymous signature mode.

In Table 5, we compare the concrete sizes of credentials and signatures, where the sizes involving n and u are the incremental sizes when the support of attributes is required. The LRSW-DAA scheme CPS has the smallest size for signatures without considering attributes, but the largest overhead to support attributes. For the SDH-DAA schemes providing a better support of attributes, BL has the smallest sizes for credentials and signatures, but has no security proof in a valid security model. In terms of the signature size, our scheme $\mathtt{DAA}_{\mathtt{OPT}}$ is the same as CDL for fully anonymous signature mode, but larger than CDL for pseudonymous signature mode. This is a trade-off from the faster signing time demonstrated in Table 4. The signature size in $\mathtt{DAA}_{\mathtt{OPT}}$ is acceptable, especially for the applications that only one signature is sent in every transaction. The applications include remote attestation, anonymous subscription services, anonymous V2X, FIDO authentication etc. In the applications, the signing time is more crucial than the signature size, as only one signature needs to be sent, where a signature in $\mathtt{DAA}_{\mathtt{OPT}}$ has at most 0.7KB/1.7KB (resp., 1KB/2.5KB) when $n = 0$ (resp., $n = 10$).

7 Conclusion

We propose the first DAA scheme with fully optimal TPM signing efficiency. We prove that our DAA scheme is secure under the DDH, DBDH and q-SDH assumptions in the UC security model [CDL16b,CDL16a] and the random oracle model. Our DAA scheme can be implemented by the TPM 2.0 commands when considering three DAA use cases. Our scheme provides significantly better signing efficiency than known DAA schemes supported by TPM 2.0. We also extend our DAA scheme to support signature-based revocation and to guarantee privacy in presence of subverted TPMs, which are presented in Appendix D and Appendix E respectively.

Acknowledgements

The authors would like to thank Jiang Zhang and anonymous reviewers for their helpful comments.

References

AKS12. Man Ho Au, Apu Kapadia, and Willy Susilo. BLACR:TTP-free blacklistable anonymous credentials with reputation. In *Proceedings of The 19th Annual Network and Distributed System Security Symposium – NDSS'12*, pages 1–17. USA: Internet Society, 2012.

ALT+15. Ghada Arfaoui, Jean-François Lalande, Jacques Traoré, Nicolas Desmoulins, Pascal Berthomé, and Saïd Gharout. A practical set-membership proof for privacy-preserving nfc mobile ticketing. *Proceedings on Privacy Enhancing Technologies*, 2015(2):25–45, 2015.

ARM. ARM Ltd. TrustZone. http://www.arm.com/products/processors/technologies/trustzone.php.

ASM06. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In *Security and Cryptography for Networks – SCN'06*, volume 4116 of *LNCS*, pages 111–125. Springer-Verlag, 2006.

BB04. Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT'04*, volume 3027 of *LNCS*, pages 223–238. Springer-Verlag, 2004.

BB08. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, 2008.

BBDT17. Amira Barki, Solenn Brunet, Nicolas Desmoulins, and Jacques Traoré. Improved algebraic MACs and practical keyed-verification anonymous credentials. In *Selected Areas in Cryptography – SAC 2016*, volume 10532 of *LNCS*, pages 360–380. Springer International Publishing, 2017.

BBS04. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO'04*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, 2004.

BCC04. Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security – CCS'04*, pages 132–145. ACM Press, 2004.

BCL08. Ernie Brickell, Liqun Chen, and Jiangtao Li. A new direct anonymous attestation scheme from bilinear maps. In *the 1st International Conference on Trusted Computing – TRUST 2008*, volume 4968 of *LNCS*, pages 166–178. Springer-Verlag, 2008.

BCL12. Ernie Brickell, Liqun Chen, and Jiangtao Li. A (corrected) DAA scheme using batch proof and verification. In *the 3rd International Conference on Trusted Systems – INTRUST'11*, volume 6151 of *LNCS*, pages 304–337. Springer-Verlag, 2012.

BCN14. Joppe W. Bos, Craig Costello, and Michael Naehrig. Exponentiating in pairing groups. In *Selected Areas in Cryptography – SAC'13*, volume 8282 of *LNCS*, pages 438–455. Springer-Verlag, 2014.

BD18. Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, pages 1–39, Jan 2018.

BFG+13. D. Bernhard, G. Fuchsbauer, E. Ghadafi, N.P. Smart, and B. Warinschi. Anonymous attestation with user-controlled linkability. *International Journal of Information Security*, 12(3):219–249, 2013.

BL07. Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 6th ACM Workshop on Privacy in the Electronic Society – WPES'07*, pages 21–30. ACM Press, 2007.

BL10a. Ernie Brickell and Jiangtao Li. Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. In *Second International Conference on Social Computing – SocialCom 2010*, pages 768–775. IEEE, 2010.

BL10b. Ernie Brickell and Jiangtao Li. A pairing-based DAA scheme further reducing TPM resources. In *Trust and Trustworthy Computing – TRUST 2010*, volume 6101 of *LNCS*, pages 181–195. Springer-Verlag, 2010.

BN06. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC'05*, volume 3897 of *LNCS*, pages 319–331. Springer-Verlag, 2006.

BP17. Alessandro Budroni and Federico Pintore. Efficient hash maps to $\mathbb{G}_2$ on BLS curves. Cryptology ePrint Archive, Report 2017/419, 2017. https://eprint.iacr.org/2017/419.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security – CCS'93*, pages 62–73. ACM Press, 1993.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science – FOCS'01*, pages 136–145, 2001. Full version is available at https://eprint.iacr.org/2000/067.

Can04. Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings of the 17th IEEE Workshop on Computer Security Foundations – CSFW'04*, page 219. IEEE Computer Society, 2004.

CCD+17. 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*, pages 901–920. IEEE, 2017.

CDD17. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to Blockchain. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security – CCS'17*, pages 683–699. ACM, 2017.

CDE+17. Jan Camenisch, Manu Drijvers, Alec Edgington, Anja Lehmann, Rolf Lindemann, and Rainer Urian. FIDO ECDAA algorithm, implementation draft. https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.pdf, February 2017.

CDH16. Jan Camenisch, Manu Drijvers, and Jan Hajny. Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In *WPES'16*, pages 123–133. ACM, 2016.

CDL16a. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong Diffie-Hellman assumption revisited. In *the 9th International Conference on Trust and Trustworthy Computing – TRUST 2016*, volume 9824 of *LNCS*, pages 1–20. Springer International Publishing, 2016. The full version is available at https://eprint.iacr.org/2016/663.

CDL16b. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In *Public-Key Cryptography – PKC 2016*, volume 9615 of *LNCS*, pages 234–264. Springer Berlin Heidelberg, 2016.

CDL17. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation with subverted TPMs. In *Advances in Cryptology – CRYPTO 2017*, volume 10403 of *LNCS*, pages 427–461. Springer, Cham, 2017.

CF08. Xiaofeng Chen and Dengguo Feng. Direct anonymous attestation for next generation TPM. *Journal of Computers*, 3(12):43–50, 2008.

Che10. Liqun Chen. A DAA scheme requiring less TPM resources. In *the 5th China International Conference on Information Security and Cryptology – Inscrypt'09*, volume 6151 of *LNCS*, pages 350–365. Springer-Verlag, 2010.

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO'04*, volume 3152 of *LNCS*, pages 56–72. Springer-Verlag, 2004.

CL13. Liqun Chen and Jiangtao Li. Flexible and scalable digital signatures in TPM 2.0. In *Proceedings of the 20th ACM Conference on Computer and Communications Security – CCS'13*, pages 37–48. ACM Press, 2013.

CLR+10. Emanuele Cesena, Hans Löhr, Gianluca Ramunno, Ahmad-Reza Sadeghi, and Davide Vernizzi. Anonymous authentication with TLS and DAA. In *Trust and Trustworthy Computing – TRUST 2010*, volume 6101 of *LNCS*, pages 47–62. Springer, 2010.

CMS08. Liqun Chen, Paul Morrissey, and Nigel P. Smart. Pairings in trusted computing. In *Pairing-Based Cryptography – Pairing 2008*, volume 5209 of *LNCS*, pages 1–17. Springer-Verlag, 2008.

CPS10. Liqun Chen, Dan Page, and Nigel P. Smart. On the design and implementation of an efficient DAA scheme. In *Smart Card Research and Advanced Application – CARDIS 2010*, volume 6035 of *LNCS*, pages 223–237. Springer-Verlag, 2010.

CPS14. Sébastien Canard, David Pointcheval, and Olivier Sanders. Efficient delegation of zero-knowledge proofs of knowledge in a pairing-friendly setting. In *Public-Key Cryptography – PKC 2014*, volume 8383 of *LNCS*, pages 167–184. Springer Berlin Heidelberg, 2014.

CS97. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology – CRYPTO'97*, volume 1296 of *LNCS*, pages 410–424. Springer-Verlag, 1997.

CS03. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer Berlin Heidelberg, 2003.

CU15. Liqun Chen and Rainer Urian. DAA-A: Direct anonymous attestation with attributes. In *Trust and Trustworthy Computing – TRUST 2015*, volume 9229 of *LNCS*, pages 228–245. Springer International Publishing, 2015.

CW10. Liqun Chen and Bogdan Warinschi. Security of the TCG privacy-CA solution. In *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 609–616, 2010.

DLST14. Nicolas Desmoulins, Roch Lescuyer, Olivier Sanders, and Jacques Traoré. Direct anonymous attestations with dependent basename opening. In *Cryptology and Network Security - CANS 2014*, volume 8813 of *LNCS*, pages 206–221. Springer International Publishing, 2014.

EGM96. Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

FCKRH12. Laura Fuentes-Castañeda, Edward Knapp, and Francisco Rodríguez-Henríquez. Faster hashing to $\mathbb{G}_2$. In *Selected Areas in Cryptography – SAC 2011*, volume 7118 of *LNCS*, pages 412–430. Springer Berlin Heidelberg, 2012.

FHKP13. Eduarda S.V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In *Public-Key Cryptography – PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer Berlin Heidelberg, 2013.

FID17. FIDO Alliances. FIDO alliance universal authentication framework (UAF) 1.1 specifications. https://fidoalliance.org/download/, February 2017.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1986.

Gal05. David Galindo. Boneh-Franklin identity based encryption revisited. In *Automata, Languages and Programming*, volume 3580 of *LNCS*, pages 791–802. Springer Berlin Heidelberg, 2005.

GPS08. S.D. Galbraith, K.G. Paterson, and N.P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

Int13. International Organization for Standardization. ISO/IEC 20008-2: Information technology - Security techniques - Anonymous digital signatures - Part 2: Mechanisms using a group public key, 2013.

ISO09. ISO/IEC 11889:2009. Information technology - Security techniques - Trusted Platform Module, 2009.

ISO15. ISO/IEC 11889:2015. Information technology - Trusted Platform Module Library, 2015.

KB16. Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, LNCS, pages 543–571. Springer Berlin Heidelberg, 2016.

KLL+18. Vireshwar Kumar, He Li, Noah Luther, Pranav Asokan, Jung-Min (Jerry) Park, Kaigui Bian, Martin B. H. Weiss, and Taieb Znati. Direct anonymous attestation with efficient verifier-local revocation for subscription system. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security – ASIACCS'18*, pages 567–574. ACM, 2018.

LDK+13. Michael Z. Lee, Alan M. Dunn, Jonathan Katz, Brent Waters, and Emmett Witchel. Anon-pass: Practical anonymous subscriptions. In *2013 IEEE Symposium on Security and Privacy*, pages 319–333. IEEE, 2013.

LRSW99. Anna Lysyanskaya, Ron Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography – SAC'99*, volume 1758 of *LNCS*, pages 184–199. Springer-Verlag, 1999.

NIS15. NISTIR 8062. Privacy risk management for federal information systems, May 2015.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer Berlin Heidelberg, 1999.

Ped92. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer-Verlag, 1992.

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

PSFK15. Jonathan Petit, Florian Schaub, Michael Feiri, and Frank Kargl. Pseudonym schemes in vehicular networks: A survey. *IEEE Communications Surveys Tutorials*, 17(1):228–255, 2015.

RSW+16. Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, Magnus Nystrom, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. fTPM: A software-only implementation of a TPM chip. In *25th USENIX Security Symposium (USENIX Security 2016)*, pages 841–856. USENIX Association, 2016.

Sch91. C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

Tru03. Trusted Computing Group. TCG TPM specification 1.2. Available at http://www.trustedcomputinggroup.org, 2003.

Tru15. Trusted Computing Group. Available at http://www.trustedcomputinggroup.org/authentication/, March 2015.

Tru16. Trusted Computing Group. Trusted platform module library specification, family "2.0" level 00 revision 01.38. Available at https://trustedcomputinggroup.org/tpm-library-specification/, September 2016.

WCG+17. Jorden Whitefield, Liqun Chen, Thanassis Giannetsos, Steve Schneider, and Helen Treharne. Privacy-enhanced capabilities for vanets using direct anonymous attestation. In *2017 IEEE Vehicular Networking Conference (VNC)*, pages 123–130, Nov 2017.

A Formal Security Model for DAA

In this section, we define the UC security model [CDL16b,CDL16a]. In UC, an environment $\mathcal{Z}$ gives inputs to the protocol parties and receives their outputs. In the real world, honest parties execute the protocol over a network controlled by an adversary $\mathcal{A}$, who may communicate freely with $\mathcal{Z}$. In the ideal world, honest parties forward their inputs to an ideal functionality $\mathcal{F}$, which then internally performs the defined task and generates the parties' outputs that are forwarded to $\mathcal{Z}$ by them.

Informally, we say that a protocol Π securely realizes an ideal functionality $\mathcal{F}$, if the real world in which Π is used is as secure as the ideal world where $\mathcal{F}$ is used. To prove the statement, one needs to show that for every adversary $\mathcal{A}$ mounting an attack in the real world, there exists an ideal world adversary (often called simulator) $\mathcal{S}$ that performs an equivalent attack in the ideal world. More precisely, Π securely realizes $\mathcal{F}$ if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$, such that no environment $\mathcal{Z}$ can distinguish interacting with the real world with Π and $\mathcal{A}$ from interacting with the ideal world with $\mathcal{F}$ and $\mathcal{S}$.

Now, we review the formal definition [CDL16b] of ideal functionality $\mathcal{F}^l_{\mathsf{daa}}$ with static corruption, meaning that the adversary decides beforehand which parties are corrupted and makes the information known to the ideal functionality. We further extend the definition to support the functionality of attributes following the modification [CDL16a].

In the UC model, different instances of the protocol are distinguished with session identifiers. Following [CDL16b], we use session identifiers of the form $sid = (\mathcal{I}, sid')$ for some issuer $\mathcal{I}$ and a unique string sid'. To allow multiple sub-sessions for the join and sign related interfaces, we use unique sub-session identifiers $jsid$ and $ssid$. $\mathcal{F}^l_{\mathsf{daa}}$ is parametrized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$, which models the information leakage that occurs in the communication between a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$. As $\mathcal{F}^l_{\mathsf{daa}}$ is extended to support attributes, we have parameters n and $\{\mathbb{A}_i\}_{1\le i\le n}$, where n is the number of attributes that every membership credential includes and $\mathbb{A}_i$ is the set from which the i-th attribute is taken. Following [CDL16a], a parameter $\mathbb{P}$ is used to describe which proofs over the attributes a platform can make. Using this generic method, the ideal functionality capture both simple protocols that only support *selective attribute disclosure* and more advanced protocols that support arbitrary predicates. Every value $\hat{p} \in \mathbb{P}$ is a predicate over the attributes, i.e., $\hat{p} : \mathbb{A}_1 \times \cdots \times \mathbb{A}_n \to \{0,1\}$.

Below, we show several algorithms ($\mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}$, $\mathsf{identify}$) which are provided by the simulator and will be used in the ideal functionality.

- $gsk \leftarrow \mathsf{ukgen}()$ will be used to generate a secret key gsk for an honest platform.
- $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ takes as input gsk, a message m, a basename bsn and a predicate $\hat{p}$, and outputs a signature σ. The algorithm will be used for honest platforms.
- $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$ takes as input a message m, a basename bsn, a signature σ and a predicate $\hat{p}$, and then outputs $f = 1$ if σ is valid on m w.r.t. bsn and $\hat{p}$ and $f = 0$ otherwise. This algorithm will be used in the VERIFY interface.
- $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$ takes as input two message/signature pairs (m_0, σ_0) and (m_1, σ_1) and a basename bsn, and outputs $f = 1$ if both signatures were created by the same platform and $f = 0$ otherwise. This algorithm will be used in the LINK interface.
- $f \leftarrow \mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk)$ takes as input a message m, a basename bsn, a signature σ and a secret key gsk, and outputs $f = 1$ if σ is a signature on m w.r.t. basename bsn under key gsk and $f = 0$ otherwise. This algorithm will allow $\mathcal{F}^l_{\mathsf{daa}}$ to perform multiple consistency checks whenever a new key gsk is created or provided by the simulator.

While algorithms ukgen and sig are *probabilistic*, the other three algorithms are *deterministic*. Besides, the link algorithm has to be *symmetric*, i.e., for all inputs it must hold that $\mathsf{link}(m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}) =$

Setup

1. `Issuer Setup`. On input (SETUP, sid) from issuer $\mathcal{I}$.
   - Verify that $sid = (\mathcal{I}, sid')$ and output (SETUP, sid) to $\mathcal{S}$.
2. `Set Algorithms`. On input $(\mathsf{ALG}, sid, \mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify})$ from $\mathcal{S}$.
   - Check that ver, link and $\mathsf{identify}$ are deterministic **(i)**.
   - Store $(sid, \mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

3. `Join Request`. On input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ from host $\mathcal{H}_j$.
   - Create a join session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
4. `Join Request Delivery`. On input $(\mathsf{JOINSTART}, sid, jsid)$ from $\mathcal{S}$.
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $status \leftarrow delivered$.
   - Abort if $\mathcal{I}$ or $\mathcal{M}_i$ is honest and a record $\langle \mathcal{M}_i, *, *, *\rangle \in$ `Members` already exists **(ii)**.
   - Output $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ to $\mathcal{I}$.
5. `Join Proceed`. On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$ with $attrs \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$.
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $\perp \leftarrow attrs$ and $status \leftarrow complete$.
   - Output $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs')$ to $\mathcal{S}$, where $attrs' \leftarrow \perp$ if $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest and $attrs' \leftarrow attrs$ otherwise.
6. `Platform Key Generation`. On input $(\mathsf{JOINCOMPLETE}, sid, jsid, gsk)$ from $\mathcal{S}$.
   - Look up record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ with $status = complete$.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, set $gsk \leftarrow \perp$.
   - Else verify that the provided gsk is eligible via checking
     - $\mathsf{CheckGskHonest}(gsk) = 1$ **(iii)** if $\mathcal{M}_i$ is honest and $\mathcal{H}_j$ is corrupted, or
     - $\mathsf{CheckGskCorrupt}(gsk) = 1$ **(iv)** if $\mathcal{M}_i$ is corrupted.
   - Add $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Memebers` and output $(\mathsf{JOINED}, sid, jsid)$ to $\mathcal{H}_j$.

Fig. 4. The Setup and Join Related Interfaces of Ideal Functionality $\mathcal{F}^l_{\mathsf{daa}}$. The roman numbers are the labels for the different checks made within the ideal functionality and will be used as reference in the security proof.

$\mathsf{link}(m_1, \sigma_1, \hat{p}_1, m_0, \sigma_0, \hat{p}_0, \mathsf{bsn})$. Note that algorithms ver and link only assist the ideal functionality for signatures which are not produced by $\mathcal{F}^l_{\mathsf{daa}}$ itself. For signatures generated by the functionality, $\mathcal{F}^l_{\mathsf{daa}}$ enforces correct verification and linkage using its internal records.

We provide the detailed definition of ideal functionality $\mathcal{F}^l_{\mathsf{daa}}$ in Figure 4 and Figure 5, and refer the reader to [CDL16b,CDL16a] for the explanations of $\mathcal{F}^l_{\mathsf{daa}}$ and the argument of why $\mathcal{F}^l_{\mathsf{daa}}$ realizes the desired security properties. The ideal functionality will use two "macros" to decide whether a key gsk is consistent with its internal records or not, where the two macros are used relying on whether a TPM is honest or corrupted. Both macros output 1 indicating a new key gsk is consistent with the internal records and 0 that signals an invalid key. The two macros are defined as below:

$$\begin{aligned}\mathsf{CheckGskHonest}(gsk) =& \forall\langle m, \mathsf{bsn}, \sigma, *, *\rangle \in \mathtt{Signed} : \mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 0 \wedge \\ & \forall\langle m, \mathsf{bsn}, \sigma, *, 1\rangle \in \mathtt{VerResults} : \mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 0\end{aligned}$$

$$\begin{aligned}&\mathsf{CheckGskCorrupt}(gsk) = \nexists m, \mathsf{bsn}, \sigma : \\ &\Big(\big(\langle m, \mathsf{bsn}, \sigma, *, *\rangle \in \mathtt{Signed} \vee \langle m, \mathsf{bsn}, \sigma, *, 1\rangle \in \mathtt{VerResults}\big) \ \wedge\ \exists gsk' : \big(gsk \neq gsk' \wedge \big(\langle *, *, gsk', *\rangle \in \\ &\mathtt{Members} \vee \langle *, *, gsk'\rangle \in \mathtt{DomainKeys}\big) \ \wedge \mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = \mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1\big)\Big).\end{aligned}$$

To simplify the definition of $\mathcal{F}^l_{\mathsf{daa}}$, the following conventions are made : 1) all requests other than the SETUP are ignored until one setup phase is completed; 2) when $\mathcal{F}^l_{\mathsf{daa}}$ performs any check that fails, it outputs $\perp$ directly to the caller; 3) whenever $\mathcal{F}^l_{\mathsf{daa}}$ runs one of the algorithms $\mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}$, it does so without maintaining states.

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ from host $\mathcal{H}_j$ with $\hat{p} \in \mathbb{P}$.
 – If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
 – Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status \leftarrow request$.
 – Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
 – Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ to $status \leftarrow delivered$.
 – Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
 – Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status = delivered$.
 – Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
 – If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
 – If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
  • If $\mathsf{bsn} \neq \perp$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ **(v)** and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle$ in `DomainKeys`.
  • Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ and check $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$ **(vi)**.
  • Check that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ **(vii)** and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$ **(viii)**.
 – If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p} \rangle$ in `Signed`.
 – Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

11. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
 – Retrieve all pairs $(\mathcal{M}_i, gsk_i)$ from $\langle \mathcal{M}_i, *, gsk_i, * \rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i \rangle \in$ `DomainKeys` such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
  • More than one key gsk_i was found **(ix)**.
  • $\mathcal{I}$ is honest and no pair $(\mathcal{M}_i, gsk_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in$ `Members` with $\hat{p}(attrs) = 1$ exists **(x)**.
  • There is an honest $\mathcal{M}_i$ but no entry $\langle m, \mathsf{bsn}, *, \mathcal{M}_i, \hat{p} \rangle \in$ `Signed` exists **(xi)**.
  • There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$ and no pair $(\mathcal{M}_i, gsk_i)$ for an honest $\mathcal{M}_i$ was found **(xii)**.
 – If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$ **(xiii)**.
 – Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

12. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
 – Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$ **(xiv)**.
 – For each key gsk_i in `Members` and `DomainKeys`, compute $b_i \leftarrow \mathsf{identify}(m_0, \mathsf{bsn}, \sigma_0, gsk_i)$ and $b_i' \leftarrow \mathsf{identify}(m_1, \mathsf{bsn}, \sigma_1, gsk_i)$, and then do the following:
  • Set $f \leftarrow 0$ if $b_i \neq b_i'$ for some i **(xv)**.
  • Set $f \leftarrow 1$ if $b_i = b_i' = 1$ for some i **(xvi)**.
 – If f is not defined yet, set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
 – Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 5. The Sign, Verify, and Link Related Interfaces of Ideal Functionality $\mathcal{F}_{\mathsf{daa}}^{l}$.

B Alternative Description of Our DAA Protocol for UC Security

In this section, we provide an alternative description of our DAA protocol $\mathtt{DAA_{OPT}}$ for UC security. We add session identifiers to $\mathtt{DAA_{OPT}}$, which is required for universal composability.

We assume that a common reference string functionality $\mathcal{F}_{\mathsf{crs}}^{D}$ and a certification authority functionality $\mathcal{F}_{\mathsf{ca}}$ are available for all parties. The former will be used to provide the parties with the system parameters params, and the latter will allow the issuer to register its public key ipk. The communication between the TPM and host is modeled using the secure message transmission functionality $\mathcal{F}_{\mathsf{smt}}^{l}$ which enables confidential and authenticated communication. In fact, $\mathcal{F}_{\mathsf{smt}}^{l}$ is naturally guaranteed by the physical proximity of the TPM and host forming a platform [CDL16b]. We refer the reader to [Can01,Can04] for the definitions of the standard ideal functionalities $\mathcal{F}_{\mathsf{crs}}^{D}$, $\mathcal{F}_{\mathsf{ca}}$ and $\mathcal{F}_{\mathsf{smt}}^{l}$. For the sake of readability, we will not explicitly write that the parties call $\mathcal{F}_{\mathsf{crs}}^{D}$ and $\mathcal{F}_{\mathsf{ca}}$ to retrieve the system parameters params and the issuer's public key ipk, nor explicitly describe that the TPM and host call $\mathcal{F}_{\mathsf{smt}}^{l}$ for communication between them, which is in line with previous work [CDL16b,CDL16a,CCD+17]. We use the ideal functionality $\mathcal{F}_{\mathsf{auth*}}$ introduced in [CDL16b] to model the semi-authenticated channel between the TPM and issuer. In particular, the TPM can use $\mathcal{F}_{\mathsf{auth*}}$ to send its public key tpk to the issuer via the host.

An alternative description of our protocol $\mathtt{DAA_{OPT}}$ for UC security is shown as follows.

Setup. On input (SETUP, sid), the issuer $\mathcal{I}$ checks that $sid = (\mathcal{I}, sid')$ for some sid', and then creates its public key $\mathsf{ipk} = (\{h_i\}_{i=0}^{n}, w, \pi_1)$ and secret key $\mathsf{isk} = \gamma$ as described in §4.2. Then $\mathcal{I}$ registers ipk with $\mathcal{F}_{\mathsf{ca}}$, and outputs $(\mathsf{SETUPDONE}, sid)$.

Join. A platform consisting of a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$ executes the join protocol with $\mathcal{I}$ as follows:

1. Upon input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$, $\mathcal{H}_j$ parses $sid = (\mathcal{I}, sid')$, and sends a message $(\mathsf{JOIN}, sid, jsid)$ to $\mathcal{I}$.
2. Upon receiving $(\mathsf{JOIN}, sid, jsid)$ from a party $\mathcal{H}_j$, $\mathcal{I}$ chooses a fresh nonce $N_I \xleftarrow{\$} \{0,1\}^{\lambda}$ and sends $(sid, jsid, N_I)$ back to $\mathcal{H}_j$.
3. Upon receiving $(sid, jsid, N_I)$ from issuer $\mathcal{I}$, $\mathcal{H}_j$ sends $(\mathtt{TPM.Create}, sid, jsid)$ to $\mathcal{M}_i$. $\mathcal{M}_i$ checks that no key record exists,[6] chooses $tsk \xleftarrow{\$} \mathbb{Z}_p$, and stores a key record $(sid, \mathcal{H}_j, tsk)$. Then $\mathcal{M}_i$ sends a TPM public key tpk back to $\mathcal{H}_j$. $\mathcal{M}_i$ and $\mathcal{H}_j$ jointly generate $\pi_t \leftarrow \mathsf{SPK}_t\{(tsk) : tpk = \bar{g}^{tsk}\}(\text{“TPM.join”}, N_I)$ via running a protocol described in Figure 2, where the only difference is that $\mathcal{H}_j$ additionally sends $(sid, jsid)$ to $\mathcal{M}_i$ when sending $\mathtt{TPM.Commit}$ or $\mathtt{TPM.Sign}$ requests.
4. $\mathcal{H}_j$ notices $\mathcal{M}_i$ sending tpk over $\mathcal{F}_{\mathsf{auth*}}$ to $\mathcal{I}$. $\mathcal{H}_j$ computes a commitment $C \leftarrow \bar{g}^{hsk} h_0^{u'}$ and a platform public key $gpk \leftarrow tpk \cdot \bar{g}^{hsk}$. Then $\mathcal{H}_j$ generates $\pi_h \leftarrow \mathsf{SPK}_h\{(hsk, u') : C = \bar{g}^{hsk} h_0^{u'}\}(\text{“Host.join”}, N_I)$ as in Figure 2.
5. $\mathcal{H}_j$ appends C, π_t, π_h to the message tpk, which is sent to $\mathcal{I}$ over $\mathcal{F}_{\mathsf{auth*}}$.
6. Upon receiving (tpk, C, π_t, π_h) from $\mathcal{F}_{\mathsf{auth*}}$ where tpk is authenticated by TPM $\mathcal{M}_i$, $\mathcal{I}$ verifies the validity of proofs π_t and π_h as in Figure 2, and checks that $\mathcal{M}_i$ did not join before. $\mathcal{I}$ stores $(jsid, tpk, C, \mathcal{M}_i, \mathcal{H}_j)$ and outputs $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$.

The join session is completed, when the issuer receives an explicit input that tells it to proceed with join session $jsid$ and issue attributes $attrs = (a_1, \ldots, a_n)$.

1. Upon input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$, $\mathcal{I}$ retrieves the record $(jsid, tpk, C, \mathcal{M}_i, \mathcal{H}_j)$ and marks $\mathcal{M}_i$ as “joined”. Then $\mathcal{I}$ creates a credential $A \leftarrow (g_1 \cdot tpk \cdot C \cdot h_0^{u''} \cdot \prod_{i=1}^{n} h_i^{a_i})^{1/(\gamma+x)}$ for two randomnesses $u'', x \in \mathbb{Z}_p$. $\mathcal{I}$ sends $(sid, jsid, (A, x, u''), attrs)$ to $\mathcal{H}_j$ over $\mathcal{F}_{\mathsf{auth*}}$. We assume that $\mathcal{F}_{\mathsf{auth*}}$ also provides the confidentiality of $((A, x, u''), attrs)$. This assumption holds when we use the method [CW10] to realize functionality $\mathcal{F}_{\mathsf{auth*}}$.[7]
2. Upon receiving $(sid, jsid, (A, x, u''), attrs)$ from issuer $\mathcal{I}$, $\mathcal{H}_j$ computes $u \leftarrow u' + u'' \mod p$ and $Y \leftarrow g_1 \cdot gpk \cdot h_0^{u} \cdot \prod_{i=1}^{n} h_i^{a_i}$, and then checks that $e(A, w \cdot g_2^{x}) = e(Y, g_2)$. Host $\mathcal{H}_j$ stores $(sid, \mathcal{M}_i, cre = (A, x, u, Y, gpk, hsk), attrs)$ and outputs $(\mathsf{JOINED}, sid, jsid)$.

Sign. The sign protocol runs between a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$. By executing the protocol, they can jointly sign a message m w.r.t. a basename bsn and attribute predicate (D, I).

---

[6] If we consider a TPM with different keys as multiple different “TPMs” with a single key, the check of key record can be omitted.

[7] As such, the DAA schemes [CDL16a,CCD+17] need to keep the credential and attributes of a platform confidential in the join protocol.

1. Upon input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, (D, I))$, $\mathcal{H}_j$ retrieves the join record $(sid, \mathcal{M}_i, cre = (A, x, u, Y, gpk, hsk), attrs)$. Then $\mathcal{H}_j$ checks if his attributes fulfill the predicate, i.e., it parses $attrs$ as $(a_1, \ldots, a_n)$ and I as $(a'_1, \ldots, a'_n)$ and checks that $a_i = a'_i$ for each $i \in D$. Next, $\mathcal{H}_j$ randomizes the credential as $T_1 \leftarrow A^{t_1}$, $Y' \leftarrow Y^{t_1} h_0^{-t_2}$ and computes $T_2 \leftarrow Y^{t_1} T_1^{-x}$. $\mathcal{H}_j$ computes an unlinkable tag $(B, K = B^{gsk})$ for a random $B \in \mathbb{G}_1^*$ if $\mathsf{bsn} = \perp$ and a pseudonym $(B = \perp, K = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^{gsk})$ otherwise using public key gpk as in Figure 3. $\mathcal{H}_j$ sends $(sid, ssid, m, \mathsf{bsn}, (D, I))$ to $\mathcal{M}_i$.
2. Upon receiving $(sid, ssid, m, \mathsf{bsn}, (D, I))$ from $\mathcal{H}_j$, $\mathcal{M}_i$ asks for permission to proceed. Then $\mathcal{M}_i$ checks that a join record $(sid, \mathcal{H}_j, tsk)$ exists, and stores $(sid, ssid, m, \mathsf{bsn}, (D, I))$ and outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, (D, I))$.

The signature is completed when $\mathcal{M}_i$ gets permission to proceed for $ssid$.

1. Upon input $(\mathsf{SIGNPROCEED}, sid, ssid)$, $\mathcal{M}_i$ retrieves a join record $(sid, \mathcal{H}_j, tsk)$ and a sign record $(sid, ssid, m, \mathsf{bsn}, (D, I))$. Then $\mathcal{M}_i$ cooperates with $\mathcal{H}_j$ to generate

$$\pi_2 \leftarrow \mathsf{SPK}_2\{(gsk, \{a_i\}_{i \in \bar{D}}, x, \tilde{u}, t_2, t_3) : g_1^{-1} \prod_{i \in D} h_i^{-a_i} = Y'^{-t_3} \bar{g}^{gsk} h_0^{\tilde{u}} \prod_{i \in \bar{D}} h_i^{a_i} \wedge$$
$$T_2/Y' = T_1^{-x} h_0^{t_2} \wedge K = B^{gsk}\}(\text{“sign”}, m, \mathsf{bsn}, D, I).$$

   This is completed via executing the sign protocol described in Figure 3, except that $\mathcal{H}_j$ additionally sends $(sid, ssid)$ to $\mathcal{M}_i$ when sending `TPM.Commit` or `TPM.Sign` requests.
2. $\mathcal{H}_j$ sets $\sigma \leftarrow (T_1, T_2, Y', B, K, \pi_2)$ and outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$.

Verify. Upon input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, (D, I), \mathtt{RL})$, a party $\mathcal{V}$ verifies the signature as follows:

1. Parse σ as $(T_1, T_2, Y', B, K, \pi_2)$.
2. Check that $B \neq 1_{\mathbb{G}_1}$ if $\mathsf{bsn} = \perp$ and $B = \perp$ otherwise. If $\mathsf{bsn} \neq \perp$, compute $B \leftarrow e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))$.
3. Check that $e(T_1, w) = e(T_2, g_2)$.
4. Verify the validity of proof π_2 on message (“sign”$, m, \mathsf{bsn}, D, I)$ following the description in §4.2.
5. For every $gsk_i \in \mathtt{RL}$, check that $K \neq B^{gsk_i}$.
6. If all the checks pass, set $f \leftarrow 1$, otherwise $f \leftarrow 0$.
7. Output $(\mathsf{VERIFIED}, sid, f)$.

Link. Upon input $(\mathsf{LINK}, sid, m_0, \sigma_0, D_0, I_0, m_1, \sigma_1, D_1, I_1, \mathsf{bsn})$ with $\mathsf{bsn} \neq \perp$, a party $\mathcal{V}$ verifies the two signatures and decides whether they are linked or not.

1. Verify that both σ_0 and σ_1 are valid with respect to $(m_0, \mathsf{bsn}, D_0, I_0)$ and $(m_1, \mathsf{bsn}, D_1, I_1)$ respectively. Output $\perp$ if one of them is not valid.
2. Parse σ_0 and σ_1 as $(T_{1,0}, T_{2,0}, Y'_0, B_0, K_0, \pi_{2,0})$ and $(T_{1,1}, T_{2,1}, Y'_1, B_1, K_1, \pi_{2,1})$.
3. If $K_0 = K_1$, set $f \leftarrow 1$, otherwise $f \leftarrow 0$.
4. Output (LINK, sid, f).

C Security Proof of Our DAA Scheme

In this section, we formally state Theorem 1, and give the proof of Theorem 1 based on the security proofs by Camenisch et al. [CDL16b,CDL16a]. As pointed out by Camenisch et al. [CCD+17], the session identifiers for UC security can be omitted, if one is only concerned with stand-alone security. Thus, the security of $\mathtt{DAA_{OPT}}$ as described in §4.2 straightforwardly follows the one of the same protocol with an addition of session identifiers as described in Appendix B, which would be proved in the following theorem.

Theorem 1. *The protocol* $\mathtt{DAA_{OPT}}$ *as described in Section B securely realizes* $\mathcal{F}^l_{\mathsf{daa}}$ *with static corruption (for any polynomial number of attributes* n, $\mathbb{A}_i = \mathbb{Z}_p$ *and selective attribute disclosure as attribute predicates* $\mathbb{P}$*) under the DBDH,* $DDH_{\mathbb{G}_1}$ *and q-SDH assumptions in the* $(\mathcal{F}_{\mathsf{auth*}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}^l_{\mathsf{smt}}, \mathcal{F}^D_{\mathsf{crs}})$*-hybrid model and the random oracle model.*

Proof. In this proof, we use $\overset{c}{\approx}$ to denote the computational indistinguishability. We also use $\mathsf{EXEC}_{\mathtt{DAA_{OPT}},\mathcal{A},\mathcal{Z}}$ to denote the real world ensemble in which environment $\mathcal{Z}$ is interacting with protocol $\mathtt{DAA_{OPT}}$ and adversary $\mathcal{A}$; $\mathsf{IDEAL}_{\mathcal{F}^l_{\mathsf{daa}},\mathcal{S},\mathcal{Z}}$ to denote the ideal world ensemble in which $\mathcal{Z}$ is interacting with ideal functionality $\mathcal{F}^l_{\mathsf{daa}}$ and simulator $\mathcal{S}$. Our proof uses the known result that the BBS+ signature scheme is EUF-CMA secure under the q-SDH assumption [ASM06,CDL16a]. Thus, we can directly reduce the security of $\mathtt{DAA_{OPT}}$ to the EUF-CMA security of the BBS+ signature scheme.

We need to prove that for every PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$, such that for every PPT environment $\mathcal{Z}$

$$\mathsf{EXEC}_{\mathtt{DAA_{OPT}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F}^l_{\mathsf{daa}},\mathcal{S},\mathcal{Z}}$$

We use a sequence of games based on the ones in [CDL16b,CDL16a] to proceed the proof, and prove that it is computationally indistinguishable between two successive games. We start with the real world protocol execution. In the next game, we construct an entity $\mathcal{C}$ who runs the real world protocol for all honest parties. Then, we split $\mathcal{C}$ into a functionality $\mathcal{F}$ and a simulator $\mathcal{S}$, where $\mathcal{F}$ receives all inputs from honest parties and sends the outputs to honest parties. We start with a "dummy functionality", then gradually change $\mathcal{F}$ and $\mathcal{S}$ accordingly, and finally end up with the full functionality $\mathcal{F}^l_{\mathsf{daa}}$ and a satisfying simulator.

Prior to describing the games, we prove that the signature proofs of knowledge SPK_1, SPK_t, SPK_h and SPK_2 are zero-knowledge by constructing a simulator and showing that the simulation is perfect unless the simulator aborts with negligible probability.

- For $\mathsf{SPK}_1\{(\gamma) : w = g_2^\gamma\}(\text{"setup"})$, a simulator Sim_1 is constructed as follows: 1) pick $c, s \xleftarrow{\$} \mathbb{Z}_p$ and compute $R \leftarrow g_2^s \cdot w^{-c}$; 2) program the random oracle such that $H_3(\text{"setup"}, g_2, w, R) = c$ and abort if encountering a collision, i.e., $H_3(\text{"setup"}, g_2, w, R)$ has already been defined; 3) output $\pi_1 \leftarrow (c, s)$.
  The simulated proof has the same distribution as the real proof unless Sim_1 aborts with probability $\leq q_{h_3}/p$ which is negligible, where q_{h_3} is the number of queries to random oracle H_3.
- For $\mathsf{SPK}_t\{(tsk) : tpk = \bar{g}^{tsk}\}(\text{"TPM.join"}, N_I)$ with an honest host, a simulator Sim'_t is constructed as follows: 1) pick $c, s \xleftarrow{\$} \mathbb{Z}_p$ and compute $E \leftarrow \bar{g}^s \cdot tpk^{-c}$; 2) make a query $(\text{"TPM.join"}, \bar{g}, tpk, E, N_I)$ to random oracle H_2 and obtain c_h; 3) choose $N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$, program random oracle such that $H_1(N_t, c_h) = c$, and abort if encountering a collision; 4) output $\pi_t \leftarrow (c, s, N_t)$.
  The simulation is perfect, unless $H_1(N_t, c_h)$ has been already defined with probability at most $q_{h_1}/2^{\ell_n} \cdot (q_{h_2}/p + q_{h_1}/p)$, where q_{h_1} is the number of H_1 queries and q_{h_2} denotes the number of H_2 queries associated with label "TPM.join".
- For $\mathsf{SPK}_t\{(tsk) : tpk = \bar{g}^{tsk}\}(msg)$ with a corrupted host, a simulator Sim''_t is constructed as follows: 1) on input a request $\mathtt{TPM.Commit}$, choose $c, s \leftarrow \mathbb{Z}_p$ and compute $E \leftarrow \bar{g}^s \cdot tpk^{-c}$, and then output E; 2) on input $(\mathtt{TPM.Sign}, msg)$, pick $N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$ and program the random oracle such that $H_1(N_t, msg) = c$ and abort if encountering a collision; 3) output (N_t, s).
  The simulation is perfect, unless $H_1(N_t, msg)$ has been defined with probability at most $q_{h_1}/2^{\ell_n}$.
- For $\mathsf{SPK}_h\{(hsk, u') : C = \bar{g}^{hsk} h_0^{u'}\}(\text{"Host.join"}, N_I)$, a simulator Sim_h is constructed as follows: 1) pick $z, \hat{s}, s' \xleftarrow{\$} \mathbb{Z}_p$ and compute $R \leftarrow \bar{g}^{\hat{s}} \cdot h_0^{s'} \cdot C^{-c}$; 2) program the random oracle such that $H_2(\text{"Host.join"}, \bar{g}, h_0, C, R, N_I) = z$ and abort if encountering a collision; 3) output $\pi_h \leftarrow (z, \hat{s}, s')$.
  The simulation is perfect, unless $H_2(\text{"Host.join"}, \bar{g}, h_0, C, R, N_I)$ has already been defined with probability at most q'_{h_2}/p, where q'_{h_2} denotes the number of H_2 queries related to label "Host.join".
- For $\mathsf{SPK}_2\Big\{(gsk, \{a_i\}_{i\in\bar{D}}, x, \tilde{u}, t_2, t_3) : g_1^{-1}\prod_{i\in D} h_i^{-a_i} = Y'^{-t_3}\bar{g}^{gsk}h_0^{\tilde{u}}\prod_{i\in\bar{D}} h_i^{a_i} \wedge T_2/Y' = T_1^{-x}h_0^{t_2} \wedge K = B^{gsk}\Big\}(\text{"sign"}, m, \mathsf{bsn}, D, I)$, a simulator Sim_2 is constructed as follows: 1) pick $c, \bar{s}, s_x, s_{\tilde{u}}, s_{t_2}, s_{t_3}, \{s_{a_i}\}_{i\in\bar{D}} \xleftarrow{\$} \mathbb{Z}_p$ and $N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$; 2) compute $R_1 \leftarrow Y'^{-s_{t_3}} \cdot \bar{g}^{\bar{s}} \cdot h_0^{s_{\tilde{u}}} \cdot \prod_{i\in\bar{D}} h_i^{s_{a_i}} \cdot g_1^c \cdot \prod_{i\in D} h_i^{c\cdot a_i}$, $R_2 \leftarrow T_1^{-s_x} \cdot h_0^{s_{t_2}} \cdot (T_2/Y')^{-c}$ and $L \leftarrow B^{\bar{s}} \cdot K^{-c}$; 3) make a query $(\text{"sign"}, \bar{g}, g_1, \{h_i\}_{i=0}^n, T_1, T_2, Y', B, K, R_1, R_2, L)$ to random oracle H_2 and get c_h as the answer; 4) program the random oracle such that $H_1(N_t, m, \mathsf{bsn}, D, I, c_h) = c$ and abort if encountering a collision; 5) output $\pi_2 \leftarrow (c, \bar{s}, s_x, s_{\tilde{u}}, s_{t_2}, s_{t_3}, \{s_{a_i}\}_{i\in\bar{D}}, N_t)$.
  The simulated proof has the same distribution as the real proof, unless $H_1(N_t, m, \mathsf{bsn}, D, I, c_h)$ has been already defined with probability $\leq q_{h_1}/2^{\ell_n}(q_{h_1}/p + q''_{h_2}/p^3)$, where q''_{h_2} is the number of queries to random oracle H_2 associated with label "sign".

By rewinding and programming the random oracles, we can construct the knowledge extractors Ext_1, Ext_t, Ext_h and Ext_2 for SPK_1, SPK_t with honest hosts, SPK_h and SPK_2 respectively.

We define all intermediate functionalities and simulators in Appendix F, and then prove that they are indistinguishable from each other by a sequence of games as follows.

Game 1. This is the real world protocol. We have Game 1 = $\mathsf{EXEC}_{\mathtt{DAA_{OPT}},\mathcal{A},\mathcal{Z}}$.

Game 2. $\mathcal{C}$ receives all inputs for honest parties and simulates the real world protocol for honest parties via simply running the protocol $\mathtt{DAA_{OPT}}$ honestly. Furthermore, $\mathcal{C}$ simulates all hybrid functionalities $\mathcal{F}_{\mathsf{auth*}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}^{l}_{\mathsf{smt}}, \mathcal{F}^{D}_{\mathsf{crs}}$ honestly.

By construction, Game 2 is equivalent to Game 1.

Game 3. Now, we split $\mathcal{C}$ into a dummy functionality $\mathcal{F}$ and a simulator $\mathcal{S}$. $\mathcal{F}$ behaves as an ideal functionality, and so the messages that it sends and receives are confidential and authenticated. Thus, the adversary $\mathcal{A}$ will not notice them. Functionality $\mathcal{F}$ receives all the inputs, and forwards them to simulator $\mathcal{S}$. $\mathcal{S}$ simulates the real world protocol $\mathtt{DAA_{OPT}}$ for all honest parties, and sends the outputs to $\mathcal{F}$, who forwards them to environment $\mathcal{Z}$. The outputs generated by the honest parities simulated by $\mathcal{S}$ are not sent anywhere, and only $\mathcal{S}$ notices them. $\mathcal{S}$ sends the equivalent outputs to $\mathcal{F}$ using an OUTPUT interface such that $\mathcal{F}$ can use the same outputs.

Game 3 is simply game 2 except for structuring differently. Thus Game 3 = Game 2.

Game 4. Now, $\mathcal{F}$ uses the procedure specified in $\mathcal{F}^{l}_{\mathsf{daa}}$ to deal with the setup related interfaces. As a result, $\mathcal{S}$ will send the algorithms $\mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}$ to $\mathcal{F}$. $\mathcal{F}$ stores the algorithms from $\mathcal{S}$, and checks whether sid is the expected form or not. For corrupt issuer, $\mathcal{S}$ can extract the issuer's secret key from SPK_1.

Note that the check of $\mathcal{F}$ for sid does not change the view of $\mathcal{Z}$, as honest issuer $\mathcal{I}$ does the same check upon receiving sid and $\mathcal{S}$ calls the SETUP interface on behalf of corrupt issuer $\mathcal{I}$. Due to the soundness of SPK_1, the view of $\mathcal{Z}$ is not changed. Thus, Game 4 $\stackrel{c}{\approx}$ Game 3.

Game 5. Now, $\mathcal{F}$ responds the queries for VERIFY and LINK interfaces using the provided algorithms ver and link, instead of forwarding them to $\mathcal{S}$. Note that $\mathcal{F}$ has not to perform the additional checks (i.e., Check **(ix)**-Check(**xi**) and Check **(xv)**-Check **(xvi)**), which will be added in later games. For Check **(xii)**, $\mathcal{F}$ rejects a signature if a matched $gsk' \in \mathtt{RL}$ is found, but does not eliminate honest TPMs from this check yet.

There are no message flows for the verify and link algorithms, and so we only need to show that the outputs are equal. The verification algorithm that $\mathcal{F}$ uses is the same as the one of real-world protocol $\mathtt{DAA_{OPT}}$, except that private key revocation check is omitted. $\mathcal{F}$ performs this revocation check separately, and thus the outputs for verify queries are equal. The real-world link algorithm outputs $\perp$ if one of two signatures is invalid. $\mathcal{F}$ does the same. The algorithm compares the equality of two pseudonyms, which is exactly what $\mathcal{F}$ does. Thus, the outputs for link queries are equal. In all, Game 5 = Game 4.

Game 6. In this game, $\mathcal{F}$ is changed to handle the join related interfaces by using the same procedure as $\mathcal{F}^{l}_{\mathsf{daa}}$, but omit the additional checks (i.e., Check **(iii)**-Check **(iv)**). If at least one of the TPM and host is honest, $\mathcal{S}$ knows the identities $\mathcal{M}$ and $\mathcal{H}$, and can correctly use them towards $\mathcal{F}$ and its simulation. If both TPM and host are corrupted but the issuer is honest, $\mathcal{S}$ cannot determine the identity of the host, since the host does not authenticate itself to the issuer in the real-world join protocol. In this case, $\mathcal{S}$ has to choose an arbitrary corrupt host $\mathcal{H}$ to invoke the JOIN interface. In the $\mathsf{JOINCOMPLETE}$ interface, $\mathcal{S}$ needs to provide the secret key of the platform gsk. When the TPM (resp., host) is honest, $\mathcal{S}$ simulates the party and knows the secret key tsk (resp., hsk). When the TPM (resp., host) is corrupted but the issuer is honest, $\mathcal{S}$ can extract the secret key tsk (resp., hsk) from the proof π_t (resp., π_h). Then $\mathcal{S}$ can compute $gsk \leftarrow tsk + hsk \bmod p$. For the case that both the host and the issuer are corrupted but the TPM is honest, $\mathcal{S}$ does not need to involve $\mathcal{F}$ and simply continues the simulation of the TPM, since $\mathcal{F}$ guarantees no security properties for the case, and the TPM does not receive inputs or send outputs in the join related interfaces.

We must guarantee $\mathcal{F}$ outputs the same values as the real-world protocol. Since the join related interfaces do not output any crypto value, but only messages like start and complete, we just need to assure that whenever the real-world protocol would reach a certain output, $\mathcal{F}$ also allows the output, and vice versa. From the real world to the functionality, this is clearly satisfied, as $\mathcal{F}$ does not perform additional checks and thus will always proceed for any input that it receives from $\mathcal{S}$. For all outputs triggered by $\mathcal{F}$, $\mathcal{S}$ has to give an explicit approval, which enables $\mathcal{S}$ to block any output by $\mathcal{F}$ if the real-world protocol would not proceed at a certain point. Thus, from the functionality to the real world, this can also be satisfied.

When both the TPM and host are corrupted but the issuer is honest, $\mathcal{S}$ uses an arbitrary corrupt host when calling the JOIN interface, which will result in a different host being stored in $\mathtt{Members}$ list of $\mathcal{F}$. However, $\mathcal{F}$ never uses the identity of the host in the case that both the TPM and host are corrupted. Although $\mathcal{F}$ sets $gsk \leftarrow \perp$ when both the TPM and host are honest, this has no impact, since the signatures are still generated by $\mathcal{S}$ and the VERIFY and LINK interfaces of $\mathcal{F}$ do not perform additional checks that make use of the internal records and secret keys.

We have to argue that $\mathcal{F}$ does not prevent an execution which was allowed in the previous game. $\mathcal{F}$ only aborts if $\mathcal{M}$ has already registered and $\mathcal{I}$ is honest. Since $\mathcal{I}$ checks whether $\mathcal{M}$ has already registered or not before outputting $\mathsf{JOINPROCEED}$ in the real-world protocol, $\mathcal{F}$ keeps consistent in Game 5 and Game 6.

If $\mathcal{S}$ can extract the secret keys from proofs π_t and π_h successfully, $\mathcal{F}$ stores the keys consistent with the real-world protocol when the TPM and host are not both honest. Furthermore, $\mathcal{S}$ can simulate the real-world protocol and keep everything in sync with $\mathcal{F}$. Due to the soundness of SPK_t and SPK_h, we have Game 6 $\stackrel{c}{\approx}$ Game 5.

Game 7. For signing with $\mathsf{bsn} = \perp$, $\mathcal{F}$ now generates signatures for honest platforms using fresh keys and the ukgen and sig algorithms defined in the setup phase. The procedure for signing with $\mathsf{bsn} \neq \perp$ has not changed. One difference is that the signature created by $\mathcal{F}$ will use a credential containing dummy attribute values for the undisclosed attributes. This change is not noticeable, since only (T_1, T_2, Y') and proof π_2 are affected. $\mathcal{F}$ use sig to generate uniformly random Y' and T_1, T_2 under the constraint that $T_2 = T_1^{\gamma}$, which have the same distribution as the elements in the signatures created by the sign protocol of $\mathtt{DAA_{OPT}}$. The proof π_2 created by $\mathcal{F}$ is indistinguishable from the one generated by the real-world sign protocol due to the zero-knowledge property of SPK_2. Besides, the unlinkable-tag/pseudonym (B, K) created by $\mathcal{F}$ has the same distribution as the one from the real-world sign protocol, since the only difference is to generate (B, K) using directly secret key gsk rather than using public key gpk.

We will use a hybrid argument to prove that environment $\mathcal{Z}$ cannot notice the change that the signatures w.r.t. $\mathsf{bsn} = \perp$ are now produced by $\mathcal{F}$ using fresh keys rather than the same key. We make this change for signing inputs with $\mathsf{bsn} = \perp$ gradually. In Game $7.k.k'$, $\mathcal{F}$ forwards all signing inputs with $\mathcal{M}_i, i > k$ to $\mathcal{S}$, who creates signatures as in Game 6. Signing inputs with $\mathcal{M}_i, i < k$ are handled by $\mathcal{F}$ via using fresh keys and the ukgen and sig algorithms. For signing inputs with $\mathcal{M}_k$, the first k' signing inputs are handled by $\mathcal{F}$, and later signing inputs will be forwarded to $\mathcal{S}$. Clearly, we have Game 7.1.0 = Game 6. Let v be the number of honest platforms and ρ_k be the number of signing inputs with $\mathcal{M}_k$ and $\mathsf{bsn} = \perp$. Clearly, we have Game $7.k.\rho_k$ = Game $7.k + 1.0$ for any $k \in \{1, \ldots, v-1\}$ and Game $7.v.\rho_v$ = Game 7. Thus, to prove that no environment can distinguish Game 7 from Game 6, it is enough to show that anyone cannot distinguish Game $7.k.k' - 1$ from Game $7.k.k'$ for any $k \in [v]$ and $k' \in [\rho_k]$.

We bound the difference between Game $7.k.k' - 1$ and Game $7.k.k'$ using a reduction from the $\mathrm{DDH}_{\mathbb{G}_1}$ assumption. In this reduction, we allow $\mathcal{S}$ and $\mathcal{F}$ to share information, since in the reduction the separation of $\mathcal{S}$ and $\mathcal{F}$ is irrelevant. $\mathcal{S}$ is given a $\mathrm{DDH}_{\mathbb{G}_1}$ instance $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_2, \bar{g}, \bar{g}^{\alpha}, \bar{g}^{\beta}, \bar{g}^{\chi})$ for unknown $\alpha, \beta \in \mathbb{Z}_p$ and aims to decide whether $\chi = \alpha\beta$ or not. We modify $\mathcal{S}$ working with $\mathcal{F}$ parametrized by k, k' to obtain an intermediate game $G^7_{k,k'}$, which is the same as Game $7.k.k' - 1$ except for the following exceptions:

- $\mathcal{S}$ picks $g_1 \xleftarrow{\$} \mathbb{G}_1^*$ and sets $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \bar{g})$ as the system parameters params, as it simulates $\mathcal{F}_{\mathsf{crs}}$.
- $\mathcal{S}$ sets $\bar{g}^{\alpha}$ as the public key tpk of TPM $\mathcal{M}_k$. $\mathcal{S}$ runs Sim'_t to generate a proof π_t in the join protocol. $\mathcal{S}$ chooses $hsk \xleftarrow{\$} \mathbb{Z}_p$ as the secret key of host $\mathcal{H}_k$.
- For the k'-th signature w.r.t. $\mathsf{bsn} = \perp$ for $\mathcal{M}_k$, we modify $\mathcal{F}$ to output a signature as follows:
  1) Choose $T_1 \xleftarrow{\$} \mathbb{G}_1^*$ and compute $T_2 \leftarrow T_1^{\gamma}$, and then pick $Y' \xleftarrow{\$} \mathbb{G}_1$.
  2) Set $B \leftarrow \bar{g}^{\beta}$ and $K \leftarrow \bar{g}^{\chi} \cdot (\bar{g}^{\beta})^{hsk}$.
  3) Send (T_1, T_2, Y', B, K) to $\mathcal{S}$, who runs Sim_2 to generate a proof π_2 and sends π_2 back to $\mathcal{F}$.
  4) Output a signature $\sigma = (T_1, T_2, Y', B, K, \pi_2)$.
- Signing queries with $\mathcal{M}_k$, which are related to $\mathsf{bsn} = \perp$ but occur after the k'-th one, or are with respect to $\mathsf{bsn} \neq \perp$, are handled by $\mathcal{S}$. To create a signature for $\mathcal{M}_k$, $\mathcal{S}$ runs Sim''_t to generate the outputs of "$\mathcal{M}_k$" in the sign protocol and executes the operations at the host side following the specification of $\mathtt{DAA_{OPT}}$.

Due to the zero-knowledge property of SPK_t, proof π_t generated by Sim'_t is computationally indistinguishable from the real proof created by the witness tsk. By the zero-knowledge property of SPK_t, we have that the signatures produced by $\mathcal{S}$ using Sim''_t are computationally indistinguishable from the ones created via executing the real-world sign protocol. The elements T_1, T_2, Y' in the k'-th signature has the same distribution as the ones generated by the sig algorithm as well as the real-world sign protocol. By the zero-knowledge property of SPK_t and SPK_2, we have that $G^7_{k,k'}$ is computationally indistinguishable to Game $7.k.k' - 1$

(resp., $7.k.k'$) if $\chi = \alpha\beta$ (resp., $\chi \xleftarrow{\$} \mathbb{Z}_p$) and thus the k'-th signing query is based on the key $gsk = \alpha + hsk$ from the join phase (resp., a fresh key). Thus, no polynomial-time distinguisher can distinguish Game $7.k.k'$ from Game $7.k.k' - 1$.

Overall, we have Game 7 $\overset{c}{\approx}$ Game 6.

Game 8. For signing with $\mathsf{bsn} \neq \perp$, $\mathcal{F}$ now generates signatures for honest platforms using fresh keys and the ukgen and sig algorithms defined in the setup phase.

Again, we will use a hybrid argument to prove that environment $\mathcal{Z}$ cannot notice the change that the signatures w.r.t. fresh basename $\mathsf{bsn} \neq \perp$ are now generated by $\mathcal{F}$ using fresh keys instead of the same key. We make this change for signing inputs with $\mathsf{bsn} \neq \perp$ gradually. In Game $8.k.k'$, $\mathcal{F}$ forwards all signing inputs with $\mathcal{M}_i, i > k$ to $\mathcal{S}$, who creates signatures as in Game 7. Signing inputs with $\mathcal{M}_i, i < k$ are handled by $\mathcal{F}$ via using fresh keys and the ukgen and sig algorithms. For signing inputs with $\mathcal{M}_k$, the first k' non-empty basenames are handled by $\mathcal{F}$, and later signing inputs will be forwarded to $\mathcal{S}$. Clearly, we have Game $8.1.0 =$ Game 7. Let v be the number of honest platforms and ρ_k be the number of different basenames with $\mathcal{M}_k$ and $\mathsf{bsn} \neq \perp$. Clearly, we have Game $8.k.\rho_k$ = Game $8.k+1.0$ for any $k \in \{1, \ldots, v-1\}$ and Game $8.v.\rho_v$ = Game 8. Thus, to prove that no environment can distinguish Game 8 from Game 7, it is enough to show that anyone cannot distinguish Game $8.k.k' - 1$ from Game $8.k.k'$ for any $k \in [v]$ and $k' \in [\rho_k]$.

We bound the difference between Game $8.k.k' - 1$ and Game $8.k.k'$ using a reduction from the DBDH assumption. $\mathcal{S}$ is given a DBDH instance $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_1^\alpha, g_2^\beta, g_1^\delta, g_2^\delta, e(g_1, g_2)^\chi)$ for unknown $\alpha, \beta, \delta \in \mathbb{Z}_p$, and aims to decide whether $\chi = \alpha\beta\delta$ or not. We modify $\mathcal{S}$ working with $\mathcal{F}$ parametrized by k, k' to obtain an intermediate game $G^8_{k,k'}$, which is the same as Game $8.k.k' - 1$ except for the following exceptions:

- $\mathcal{S}$ sets $\bar{g} = g_1^\delta$ and $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \bar{g})$ as the system parameters params, as it simulates $\mathcal{F}_{\mathsf{crs}}$.
- $\mathcal{S}$ sets the unknown discrete logarithm α as the key gsk for the honest platform with $\mathcal{M}_k$. $\mathcal{S}$ chooses $C \xleftarrow{\$} \mathbb{G}_1$ and runs Sim_h to generate a proof π_h in the join protocol. $\mathcal{S}$ simulates the TPM "$\mathcal{M}_k$" honestly, i.e., choosing a key $tsk \xleftarrow{\$} \mathbb{Z}_p$ and generates SPK_t with witness tsk etc. Note that the platform public key $gpk = g_1^{\alpha\delta}$ which is unknown for $\mathcal{S}$.
- $\mathcal{S}$ chooses $j^* \xleftarrow{\$} [Q]$ as the guess that the j^*-th query bsn_{j^*} to random oracle $H_{\mathbb{G}_2}$ is used as the k'-th basename bsn^* in the signing queries, where Q is the number of $H_{\mathbb{G}_2}$ queries. Without loss of generality, we assume that $\mathcal{S}$ guesses correctly with probability $1/Q$.
- $\mathcal{S}$ maintains a $H_{\mathbb{G}_2}$-List which is initially empty. For the j-th query bsn_j to random oracle $H_{\mathbb{G}_2}$, $\mathcal{S}$ responds as follows:
  - If bsn_j has already been queried, retrieve $(\mathsf{bsn}_j, W_j, *)$ from $H_{\mathbb{G}_2}$-List and return W_j.
  - Otherwise, if $j = j^*$, respond with g_2^β and adds $(\mathsf{bsn}_{j^*}, g_2^\beta, -)$ to $H_{\mathbb{G}_2}$-List.
  - If $j \neq j^*$, pick $r \xleftarrow{\$} \mathbb{Z}_p$, compute $W_j \leftarrow g_2^r$, adds (bsn_j, W_j, r) to $H_{\mathbb{G}_2}$-List, and respond with W_j.
- When generating signatures w.r.t. the k'-th basename bsn^* for $\mathcal{M}_k$, we modify $\mathcal{F}$ to output a signature as follows:
  1) Choose $T_1 \xleftarrow{\$} \mathbb{G}_1^*$ and compute $T_2 \leftarrow T_1^\gamma$, and then pick $Y' \xleftarrow{\$} \mathbb{G}_1$.
  2) Set $B \leftarrow \perp$ and $K \leftarrow e(g_1, g_2)^\chi$.
  3) Send (T_1, T_2, Y', B, K) to $\mathcal{S}$, who runs Sim_2 to generate a proof π_2 and sends π_2 back to $\mathcal{F}$.
  4) Output a signature $\sigma = (T_1, T_2, Y', B, K, \pi_2)$.
- Signing queries with $\mathcal{M}_k$ and later basenames are handled by $\mathcal{S}$. To generate a signature w.r.t. $\mathsf{bsn} \neq \perp$, $\mathcal{S}$ does the following:
  1) Choose $T_1 \xleftarrow{\$} \mathbb{G}_1^*$ and compute $T_2 \leftarrow T_1^\gamma$, and then pick $Y' \xleftarrow{\$} \mathbb{G}_1$.
  2) Set $B \leftarrow \perp$ and retrieve $(\mathsf{bsn}, *, r)$ from $H_{\mathbb{G}_2}$-List.
  3) Compute a pseudonym $K \leftarrow e(g_1^\alpha, (g_2^\delta)^r)$. Thus, $K = e(g_1^\delta, g_2^r)^\alpha = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^{gsk}$.
  4) Run Sim_2 to generate a proof π_2 and output $\sigma = (T_1, T_2, Y', B, K, \pi_2)$.

By the zero-knowledge property of SPK_h, proof π_h generated by Sim_h is computationally indistinguishable from the real proof generated in the real-world join protocol. The elements T_1, T_2, Y' has the same distribution as the ones generated by the sig algorithm as well as the real-world sign protocol. Due to the zero-knowledge property of SPK_2, proof π_2 created by Sim_2 is computationally indistinguishable from the one generated with the witnesses. If $\mathcal{S}$ guesses correctly, $H_{\mathbb{G}_2}(\mathsf{bsn}^*) = g_2^\beta$, and thus $K = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}^*))^{gsk} = e(g_1, g_2)^{\alpha\beta\delta}$ will be a pseudonym computed in the real-world sign protocol.

Thus, we have that $G^8_{k,k'}$ is computationally indistinguishable to Game $8.k.k' - 1$ (resp., Game $8.k.k'$) if $\chi = \alpha\beta\delta$ (resp., $\chi \xleftarrow{\$} \mathbb{Z}_p$) and thus signatures with the k'-th basename bsn^* are based on the key $gsk = \alpha$

from the join phase (resp., a fresh key). Thus, no polynomial-time distinguisher can distinguish Game $8.k.k'$ from Game $8.k.k'-1$. In all, Game 8 $\stackrel{c}{\approx}$ Game 7.

Game 9. Now, $\mathcal{F}$ checks whether the platform's attributes fulfill the attribute predicate or not when the host is honest, and no longer reveals $(m, \mathsf{bsn}, \hat{p})$ to $\mathcal{S}$, but only the leakage $l(m, \mathsf{bsn}, \hat{p})$. All the adversary notices are the leakage of the secure channel between the TPM and host. $\mathcal{S}$ can still simulate this by taking dummy messages, basenames and attribute predicates that result in the same leakage and using the values to simulate the real-world protocol.

In the real-world sign protocol, the host checks if his attributes fulfill the given attribute predicate. $\mathcal{F}$ does the same check in the SIGN interface. Thus, no adversary can notice the change for $\mathcal{F}$. As simulator $\mathcal{S}$ guarantees that the dummy attribute predicate still holds for the platform's attributes, any signing query that would previously succeed will still succeed. Thus, we have Game 9 = Game 8.

Game 10. $\mathcal{F}$ no longer informs the simulator about the attributes of an honest platform in the join phase. $\mathcal{S}$ now uses dummy attributes in the join protocol. Moreover, $\mathcal{F}$ now only allows platforms that joined with attributes fulfilling the attribute predicate to sign, when $\mathcal{I}$ is honest (denoting this check by `joinatt`).

Although dummy attributes are used by $\mathcal{S}$ in the join protocol, this does not change the view of the adversary, as the credential and attributes of an honest platform are sent confidentially over $\mathcal{F}_{\mathsf{auth}^*}$ by using an encryption scheme. Functionality $\mathcal{F}$ checks whether the attribute predicate holds for the platform's attributes, and only then will $\mathcal{S}$ be notified. Thus, $\mathcal{S}$ knows that it has to simulate with some dummy attribute predicate that holds for the dummy attributes that it chooses in the join protocol. When both the TPM and host are honest, a signature is generated by $\mathcal{F}$, and thus contains the correct attributes and attribute predicate.

We show that this check `joinatt` does not change the view of environment $\mathcal{Z}$. Before signing with TPM $\mathcal{M}_i$ in the real-world protocol, an honest host $\mathcal{H}_j$ always checks whether it has joined with $\mathcal{M}_i$ and aborts otherwise. So there is no difference for honest hosts. An honest TPM $\mathcal{M}_i$ only signs, if it has joined with some host $\mathcal{H}_j$. Thus, there is no difference for honest TPMs. When an honest TPM $\mathcal{M}_i$ executes the join protocol with a corrupted host $\mathcal{H}_j$ and the honest issuer $\mathcal{I}$, $\mathcal{S}$ will make a join query with $\mathcal{F}$ on behalf of $\mathcal{H}_j$, which guarantees that $\mathcal{M}_i$ and $\mathcal{H}_j$ are in list `Members`. Thus, $\mathcal{F}$ still allows any signing that could take place in the real sign protocol.

Overall, we have Game 10 = Game 9.

Game 11. In this game, $\mathcal{F}$ additionally checks the validity of every new key gsk, which is received in the join interface or generated in the sign interface, i.e., Check **(iii)**, Check **(iv)** and Check **(v)**.

We show that these checks will fail with negligible probability. We only consider valid signatures from `VerResults` and `Signed`, where list `Signed` only contains valid signatures added for honest TPMs and hosts, and $\perp$ added for honest TPMs and corrupt hosts. Note that $\mathsf{identify}(m, \mathsf{bsn}, \perp, gsk) = 0$. Thus, we only need to consider valid signatures.

When the TPM is corrupted, $\mathcal{F}$ checks that $\mathsf{CheckGskCorrupt}(gsk) = 1$ for the key gsk which is obtained by combining the key tsk extracted from proof π_t with the key hsk extracted from proof π_h. This check prevents the adversary $\mathcal{A}$ from choosing a key $gsk \neq gsk'$ such that both keys fit to the same signature. It is impossible, since there exists only a single key gsk for each valid signature such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$, where $B \neq 1_{\mathbb{G}_1}$ if $\mathsf{bsn} = \perp$ and $H_{\mathbb{G}_2}(\mathsf{bsn}) \neq 1_{\mathbb{G}_2}$ with overwhelming probability otherwise. Thus, this check will fail with only negligible probability.

When the TPM is honest, $\mathcal{F}$ checks that $\mathsf{CheckGskHonest}(gsk) = 1$ whenever it receives or creates a new key gsk. If the host is corrupted, $\mathcal{S}$ extracts a key hsk from proof π_h and adds this key to a simulated key tsk such that obtaining the key gsk. By this check, we avoid the registration of platform keys such that matching signatures already exist. Again, there is one unique key gsk matching a valid signature as $B \neq 1_{\mathbb{G}_1}$ if $\mathsf{bsn} = \perp$ and $H_{\mathbb{G}_2}(\mathsf{bsn}) \neq 1_{\mathbb{G}_2}$ with overwhelming probability otherwise. Moreover, a key gsk chosen by the ukgen algorithm is uniformly random in an exponentially large group $\mathbb{Z}_p$, and this also holds for a simulated key tsk (and thus gsk). Thus, the probability that there already is a signature under the key gsk is negligible.

In all, we have Game 11 $\approx$ Game 10.

Game 12. Now, after creating a signature, $\mathcal{F}$ additionally checks whether the signature passes the verification and matches the correct key, i.e., Check **(vi)** and Check **(vii)**. Besides, with the help of internal key records `Members` and `DomainKeys`, $\mathcal{F}$ checks that no platform has already a key matching the newly generated signature, i.e., Check **(viii)**.

Check **(vi)** will always succeed, since the sig algorithm generates valid signatures. $\mathcal{F}$ runs the sig algorithm to set $K = B^{gsk}$ for either a random $B \in \mathbb{G}_1^*$ or $B = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))$, and thus Check **(vii)** will also always succeed.

We reduce that Check **(viii)** fails to the Discrete- Logarithm (DL) assumption in $\mathbb{G}_1$, which is implied by the assumptions claimed in Theorem 1. $\mathcal{S}$ is given a DL instance $(\bar{g}, \bar{g}^{\alpha})$ in $\mathbb{G}_1$ and attempts to output α. $\mathcal{F}$ working with $\mathcal{S}$ chooses one of signing queries with honest platforms at random, as there are only polynomial many signing queries. For this chosen signing query, $\mathcal{F}$ sharing information with $\mathcal{S}$ does the following:

1) Set the unknown α as the key gsk and $\bar{g}^{\alpha}$ as gpk.
2) Run the sig algorithm to create a signature σ with the only difference that using Sim_2 to simulate a proof π_2.
3) Output a signature σ.

When $\mathcal{F}$ re-uses the unknown key α, it repeats the above same procedure. By the zero-knowledge property of SPK_2, π_2 generated by Sim_2 is computationally indistinguishable from the ones created by the sig algorithm. Since $B \neq 1_{\mathbb{G}_1}$ for valid signatures and $H_{\mathbb{G}_2}(\mathsf{bsn}) = 1_{\mathbb{G}_2}$ with negligible probability, there is one unique key matching a valid signature with overwhelming probability. If $\mathcal{F}$ finds a key gsk matching any of the signatures created by the above process in `Members` or `DomainKeys`, it must be the discrete logarithm α, and $\mathcal{S}$ outputs gsk.

Overall, we have Game 12 $\overset{c}{\approx}$ Game 11.

Game 13. In the VERIFY interface, $\mathcal{F}$ now additionally checks whether it finds multiple platform keys identifying this signature, i.e., Check **(ix)**. If so, $\mathcal{F}$ rejects the signature.

We show that this check does not change the outputs of the VERIFY interface, since any signature that would pass the verification in Game 12 will still pass the verification in this game with overwhelming probability. If a signature σ on m w.r.t. bsn and $\hat{p}$ would pass the verification in the previous game, we have $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$. Thus, $B \neq 1_{\mathbb{G}_1}$ when $\mathsf{bsn} = \perp$. The probability that $B = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn})) = 1_{\mathbb{G}_T}$ is negligible, as $H_{\mathbb{G}_2}$ is a random oracle. A key gsk matching a signature means that $K = B^{gsk}$, and there is only a single key matching the signature when $B \neq 1$. Therefore, the event that multiple keys match a valid signature only occurs if $e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn})) = 1_{\mathbb{G}_T}$ that happens with negligible probability. Thus, Game 13 $\approx$ Game 12.

Game 14. If $\mathcal{I}$ is honest, $\mathcal{F}$ now only accepts signatures on platform keys and attribute values on which $\mathcal{I}$ issued credentials.

This check changes the verification outcome with negligible probability under the assumption that the BBS+ signature is EUF-CMA secure. This assumption holds under the q-SDH assumption [ASM06,CDL16a]. $\mathcal{S}$ is given a BBS+ public key $(\{h_i\}_{i=0}^n, w)$. In the following reduction, $\mathcal{F}$ and $\mathcal{S}$ share information, and behave exactly as in Game 14 with the following exceptions:

- $\mathcal{S}$ runs Sim_1 to generate a proof π_1 and registers a public key $(\{h_i\}_{i=0}^n, w, \pi_1)$.
- When $\mathcal{I}$ needs to issue a credential in the join protocol, $\mathcal{S}$ runs Ext_t to extract a TPM key tsk from proof π_t if the TPM is corrupted, and runs Ext_h to extract a witness (hsk, u') from proof π_h if the host is corrupted. If the TPM or the host are honest, $\mathcal{S}$ knows the related key as it simulates the party. Then, $\mathcal{S}$ computes $gsk \leftarrow tsk + hsk \mod p$. Next, $\mathcal{S}$ makes a query $(gsk, attrs)$ to its signing oracle and receives a BBS+ signature (A, x, u). Finally, $\mathcal{S}$ calculates $u'' \leftarrow u - u' \mod p$ and sends $(A, x, u'', attrs)$ on behalf of $\mathcal{I}$ over $\mathcal{F}_{\mathsf{auth}^*}$.
- When signing for honest platforms, $\mathcal{F}$ uses the signing oracle to generate BBS+ signatures on fresh keys and attributes that the platform joined with. Note that all the platform keys queried to the signing oracle are stored in `Members` or `DomainKeys`, and the attributes of platforms are stored in `Members`.
- When $\mathcal{F}$ finds a valid signature $\sigma = (T_1, T_2, Y', B, K, \pi_2)$ w.r.t. attribute predicate $\hat{p} = (D, I)$ such that no matching key gsk has been found for a certain platform with attributes fulfilling $\hat{p}$, $\mathcal{S}$ uses Ext_2 to extract a witness $(gsk, \{a_i\}_{i \in \bar{D}}, x, \tilde{u}, t_2, t_3)$ from proof π_2 such that $g_1^{-1} \prod_{i \in D} h_i^{-a_i} = Y'^{-t_3} \bar{g}^{gsk} h_0^{\tilde{u}} \prod_{i \in \bar{D}} h_i^{a_i}$, $T_2/Y' = T_1^{-x} h_0^{t_2}$ and $K = B^{gsk}$. Then $\mathcal{S}$ sets $attrs$ according the attribute disclosure (D, I) and the extracted attributes $\{a_i\}_{i \in \bar{D}}$, and computes $A \leftarrow T_1^{t_3}$ and $u \leftarrow \tilde{u} + t_2 \cdot t_3 \mod p$. $\mathcal{S}$ outputs $((gsk, attrs), (A, x, u))$ as a forgery of the BBS+ signature scheme.

Due to the zero-knowledge property of SPK_1, environment $\mathcal{Z}$ cannot distinguish a simulated proof π_1 from a real proof. By the soundness of SPK_t and SPK_h, $\mathcal{S}$ can simulate the executions of join protocol successfully. Below, we show that the extracted credential (A, x, u) is a valid BBS+ signature on a message $(gsk, attrs)$.

Since σ is a valid signature, the equation $e(T_1, w) = e(T_2, g_2)$ holds, and thus $T_2 = T_1^{\gamma}$. From the equalities $g_1^{-1} \prod_{i \in D} h_i^{-a_i} = Y'^{-t_3} \bar{g}^{gsk} h_0^{\tilde{u}} \prod_{i \in \bar{D}} h_i^{a_i}$ and $T_2/Y' = T_1^{-x} h_0^{t_2}$, we have the following relation holds: $T_1^{t_3 x} T_2^{t_3} = g_1 \bar{g}^{gsk} h_0^{\tilde{u}+t_2 t_3} \prod_{i=1}^{n} h_i^{a_i}$. Replacing $T_1^{t_3}$, T_2 and $\tilde{u} + t_2 t_3$ with A, T_1^{γ} and u respectively, we have $A^{\gamma+x} = g_1 \bar{g}^{gsk} h_0^{u} \prod_{i=1}^{n} h_i^{a_i}$. Since no matching key gsk has been found for a certain platform with attributes fulfilling (D, I), $\mathcal{F}$ and $\mathcal{S}$ never make a query $(gsk, attrs)$ to the signing oracle. Overall, we have Game 14 $\overset{c}{\approx}$ Game 13.

Game 15. Now, $\mathcal{F}$ rejects any signature σ on message m w.r.t. basename bsn and predicate $\hat{p}$ such that σ matches the key gsk of a platform with an honest TPM, but the TPM never signed m w.r.t. bsn and $\hat{p}$, i.e., additionally performing Check **(xi)**.

We use a hybrid argument to prove that environment $\mathcal{Z}$ cannot notice this change under the DL assumption. We distinguish two cases depending whether the host is honest or not.

For the case that the TPM is honest but the host is corrupt, we proceed the following hybrid argument. Game 15.i is the same as Game 14, except that $\mathcal{F}$ performs this check **(xi)** for the first i platforms with an honest TPM and a corrupt host. We use a reduction from the DL assumption to bound the difference between Game 15.$i-1$ and Game 15.i. $\mathcal{S}$ is given a DL instance $(\bar{g}, \bar{g}^{\alpha})$ in $\mathbb{G}_1$ and simulates as follows:

- $\mathcal{S}$ sets the unknown α and $\bar{g}^{\alpha}$ as the secret key tsk and respective public key tpk of TPM $\mathcal{M}_i$.
- For the join session and sign sessions with $\mathcal{M}_i$, $\mathcal{S}$ uses Sim''_t to generate the proofs of SPK_t.
- As the corresponding host $\mathcal{H}_j$ is corrupted, $\mathcal{S}$ uses Ext_h to extract hsk from proof π_h. Then, $\mathcal{S}$ computes $gpk \leftarrow tpk \cdot \bar{g}^{hsk}$ as the public key of the platform.
- For any verification query with a signature σ w.r.t. $\mathsf{bsn} \neq \perp$, $\mathcal{F}$ can check that $K = e(gpk, H_{\mathbb{G}_2}(\mathsf{bsn}))$ to decide if σ matches the key $gsk = \alpha + hsk$. When $\mathcal{F}$ finds a valid signature σ on message m w.r.t. basename $\mathsf{bsn} \neq \perp$ and attribute predicate $\hat{p}$ matching key gsk but $\mathcal{M}_i$ never signed m w.r.t. bsn and $\hat{p}$, $\mathcal{S}$ runs Ext_2 to extract key gsk from proof π_2 in signature σ. Then $\mathcal{S}$ outputs $\alpha \leftarrow gsk - hsk \mod p$ as the solution of the DL problem.
- For verification queries with signatures w.r.t. $\mathsf{bsn} = \perp$, $\mathcal{F}$ now skips the check that one pair $(\mathcal{M}_i, gsk)$ is found, as it does not know the key gsk. Since there are only polynomial many verification queries, $\mathcal{F}$ chooses one verification query at random as the guess that this is the first verification query with a signature σ on message m w.r.t. $\mathsf{bsn} = \perp$ and predicate $\hat{p}$ such that σ is valid and matches the key gsk but $\mathcal{M}_i$ never signed m w.r.t. bsn and $\hat{p}$. If $\mathcal{F}$ guesses successfully, $\mathcal{S}$ uses Ext_2 to extract the key gsk from the proof π_2 in signature σ. Then $\mathcal{S}$ outputs $\alpha \leftarrow gsk - hsk \mod p$ as the solution of the DL problem.

For the case that both the TPM and host are honest, we use a reduction from the DL assumption to bound the difference between Game 14 and Game 15. $\mathcal{S}$ is given a DL instance $(\bar{g}, \bar{g}^{\alpha})$ in $\mathbb{G}_1$, shares information with $\mathcal{F}$, and simulates as follows:

- Whenever $\mathcal{F}$ would choose a new key gsk_i to sign for an honest platform, $\mathcal{F}$ picks $r_i \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ and sets the unknown αr_i as gsk_i and computes $gpk_i \leftarrow (\bar{g}^{\alpha})^{r_i}$. Then, $\mathcal{F}$ generates a signature using the sig algorithm and public key gpk_i, except for using Sim_2 to simulate a proof π_2.
- For any verification query with a signature σ w.r.t. $\mathsf{bsn} \neq \perp$, $\mathcal{F}$ can check that $K = e(gpk_i, H_{\mathbb{G}_2}(\mathsf{bsn}))$ to decide if σ matches the key gsk_i. When $\mathcal{F}$ finds a valid signature σ on message m w.r.t. $\mathsf{bsn} \neq \perp$ and $\hat{p}$ matching some key gsk_i but the platform never signed m w.r.t. bsn and $\hat{p}$, $\mathcal{S}$ runs Ext_2 to extract key gsk_i from proof π_2 in signature σ, and outputs $gsk_i/r_i \mod p$ as the solution of the DL problem.
- For verification queries with signatures w.r.t. $\mathsf{bsn} = \perp$, $\mathcal{F}$ now skips the check that one pair $(*, gsk_i)$ for an honest platform is found, as it cannot know the key $gsk_i = \alpha r_i$. Since there are only polynomial many verification queries, $\mathcal{F}$ chooses one verification query at random as the guess that the signature σ on message m w.r.t. $\mathsf{bsn} = \perp$ and $\hat{p}$ in this query is the first valid signature such that matching some key gsk_i for an honest platform but the platform never signed m w.r.t. $\mathsf{bsn} = \perp$ and $\hat{p}$. If $\mathcal{F}$ guesses correctly, $\mathcal{S}$ uses Ext_2 to extract the key gsk_i from the proof π_2 in signature σ, and outputs $gsk_i/r_i \mod p$ as the solution of the DL problem.

By the zero-knowledge of SPK_t and SPK_2 and the soundness of SPK_h and SPK_2, Game 15 $\overset{c}{\approx}$ Game 14.

Game 16. Now $\mathcal{F}$ prevents private key revocation of platforms with an honest TPM.

If an environment $\mathcal{Z}$ can put a key gsk into the revocation list `RL` such that gsk matches a signature from a platform with an honest TPM, we can construct an algorithm breaking the DL assumption. We show this in two steps: first $\mathcal{F}$ prevents this for pairs $(\mathcal{M}_i, gsk)$ from `Members`; and then $\mathcal{F}$ prevents this also for pairs $(\mathcal{M}_i, gsk)$ from `DomainKeys`. Note that for honest platforms there are only pairs $(\mathcal{M}_i, gsk)$ in `DomainKeys` such that $gsk \neq \perp$; for honest TPMs with corrupt hosts, there are only pairs $(\mathcal{M}_i, gsk)$ in `Members`.

For the case that this check aborts for a pair found in `Members`, we can solve the DL problem. $\mathcal{S}$ is given a DL instance $(\bar{g}, \bar{g}^{\alpha})$ in $\mathbb{G}_1$. $\mathcal{S}$ chooses one platform with honest TPM $\mathcal{M}_i$ and corrupt host $\mathcal{H}_j$ at random as the guess that $(\mathcal{M}_i, *)$ is the first pair such that this check aborts. $\mathcal{S}$ sets $\bar{g}^{\alpha}$ as the public key tpk of $\mathcal{M}_i$, and extracts hsk from proof π_h. $\mathcal{S}$ uses Sim''_t to simulate the proofs of SPK_t in the join and sign protocols for $\mathcal{M}_i$. When $\mathcal{F}$ finds a key gsk in the revocation list `RL` matching a signature from the platform with honest $\mathcal{M}_i$, $\mathcal{S}$ outputs $gsk - hsk \mod p$ as the solution of the DL problem, since there is only one key matching a signature.

For the case that this check aborts for a pair found in `DomainKeys`, we can solve the DL problem. $\mathcal{S}$ is given a DL instance $(\bar{g}, \bar{g}^{\alpha})$ in $\mathbb{G}_1$. Whenever $\mathcal{F}$ would choose a new key gsk_i to sign for an honest platform, $\mathcal{F}$ picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets the unknown αr_i as gsk_i and computes $gpk_i \leftarrow (\bar{g}^{\alpha})^{r_i}$. Then, $\mathcal{F}$ generates a signature using the sig algorithm and public key gpk_i, except for using Sim_2 to simulate a proof π_2. When $\mathcal{F}$ finds a key gsk matching one signature created by gsk_i in the revocation list `RL`, $\mathcal{S}$ outputs $gsk/r_i \mod p$ as the solution of the DL problem.

In all, we have Game 16 $\overset{c}{\approx}$ Game 15.

Game 17. $\mathcal{F}$ now performs all the additional checks done by $\mathcal{F}^l_{\mathsf{daa}}$ for the LINK interface, i.e., Check **(xv)** and Check **(xvi)**.

We show that these checks do not change the output of the link queries. If a platform key matching one of two signatures but not the other, $\mathcal{F}$ outputs $f = 0$. If one key matches both signatures, $\mathcal{F}$ outputs $f = 1$. For the signatures that have already been verified, we have $B \neq 1_{\mathbb{G}_1}$ if $\mathsf{bsn} = \perp$ and $e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn})) \neq 1_{\mathbb{G}_T}$ with overwhelming probability otherwise. Thus, there is one unique key $gsk \in \mathbb{Z}_p$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ with overwhelming probability. If there is a key gsk that matches one of two signatures but not the other, we have $K_0 \neq K_1$ and the link algorithm would also output 0 by the soundness of SPK_2. If there is some key gsk matching both signatures, we have $K_0 = K_1$ and the link algorithm would also output 1 by the soundness of SPK_2.

Overall, we have Game 17 $\overset{c}{\approx}$ Game 16.

The functionality in Game 17 is equal to $\mathcal{F}^l_{\mathsf{daa}}$, i.e., Game 17 = $\mathsf{IDEAL}_{\mathcal{F}^l_{\mathsf{daa}}, \mathcal{S}, \mathcal{Z}}$, which completes our proof. □

D Our DAA Scheme with Signature-Based Revocation Extension

In this section, we extend our scheme $\mathtt{DAA_{OPT}}$ to support signature-based revocation. Our DAA scheme with signature-based revocation keeps compatible with the TPM 2.0 specification [Tru16]. While known DAA schemes with signature-based revocation [BL07,BL10a,CDL16a,CCD+17] require at least $3n_r E_{\mathbb{G}_1}$ for the TPM to prove that the platform has not been revoked, our DAA scheme provides the fully optimal TPM signing efficiency, where n_r denotes the number of revoked platforms.

We present a signature-based revocation mechanism, following the basic revocation idea in the EPID scheme [BL07]. We propose an efficient method to delegate most computations of the TPM to its host and keep the fully optimal signing efficiency for the TPM.

Now, we use $K = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}))^{gsk}$ for both $\mathsf{bsn} \neq \perp$ and $\mathsf{bsn} = \perp$, where $\mathsf{str} = \mathsf{bsn}$ if $\mathsf{bsn} \neq \perp$ and $\mathsf{str} \xleftarrow{\$} \{0,1\}^{\ell_r}$ otherwise. A verifier $\mathcal{V}$ locally maintains a signature revocation list $\mathtt{SRL} = \{(\mathsf{str}_i, K_i)\}_{i=1}^{n_r}$, where $K_i = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}_i))^{gsk_i}$ for some $gsk_i \in \mathbb{Z}_p$. To prove non-revocation towards $\mathcal{V}$, a platform with secret key gsk needs to prove in zero-knowledge $K_i \neq e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}_i))^{gsk}$ for $\forall i \in [n_r]$. The proof can be done using the zero-knowledge proof of inequality of discrete logarithms by Camenisch and Shoup [CS03]: choose $v_i \xleftarrow{\$} \mathbb{Z}_p$ and compute $V_i \leftarrow (e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}_i))^{gsk}/K_i)^{v_i}$; and then generate the following proof of knowledge:

$$\begin{aligned}\mathsf{SPK}_r \big\{ (\{v_i \cdot gsk, v_i\}_{i=1}^{n_r}) : V_i &= e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}_i))^{v_i \cdot gsk} \cdot K_i^{-v_i} \\ \wedge\ 1_{\mathbb{G}_T} &= e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}))^{v_i \cdot gsk} \cdot K^{-v_i} \text{ for each } i \in [n_r] \big\},\end{aligned}$$

where $K = e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}))^{gsk}$.

We can extend $\mathtt{DAA_{OPT}}$ to support signature-based revocation by extending the signing operations of the host and the verification algorithm, where the operations executing by the TPM keep unchanged. Concretely, the host $\mathcal{H}_j$ is further given a signature revocation list `SRL`, and additionally performing the following operations in the sign protocol to prove that the platform has not been revoked.

1. For each $i \in [n_r]$, $\mathcal{H}_j$ chooses $v_i \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $V_i \leftarrow e(gpk^{v_i}, H_{\mathbb{G}_2}(\mathsf{str}_i)) \cdot K_i^{-v_i}$. $\mathcal{H}_j$ sends a request `TPM.Commit` to the TPM and receives E as response.
2. If $\mathsf{bsn} \neq \perp$, $\mathcal{H}_j$ sets $\mathsf{str} \leftarrow \mathsf{bsn}$ and $B \leftarrow \perp$, and computes $K \leftarrow e(gpk, H_{\mathbb{G}_2}(\mathsf{bsn}))$. If $\mathsf{bsn} = \perp$, $\mathcal{H}_j$ changes the computational manner of unlinkable tags as: pick $\mathsf{str} \xleftarrow{\$} \{0,1\}^{\ell_r}$, and set $B \leftarrow \perp$ and compute $K \leftarrow e(gpk, H_{\mathbb{G}_2}(\mathsf{str}))$.
3. For each $i \in [n_r]$, $\mathcal{H}_j$ picks $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $F_i \leftarrow \tilde{E}^{v_i} \cdot \bar{g}^{\alpha_i}$; and then calculates $W_i \leftarrow e(F_i, H_{\mathbb{G}_2}(\mathsf{str}_i)) \cdot K_i^{-\beta_i}$ and $Z_i \leftarrow e(F_i, H_{\mathbb{G}_2}(\mathsf{str})) \cdot K^{-\beta_i}$, where $\tilde{E} = E\bar{g}^{\hat{r}}$.
4. $\mathcal{H}_j$ computes $c_h \leftarrow H_2(\text{"sign"}, \bar{g}, g_1, \{h_i\}_{i=0}^n, T_1, T_2, Y', B, K, R_1, R_2, L, \mathtt{SRL}, \{V_i, W_i, Z_i\}_{i=1}^{n_r})$. Then $\mathcal{H}_j$ sends $\mathtt{TPM.Sign}, (m, \mathsf{str}, D, I, c_h)$ to the TPM and receives (N_t, s) as response.
5. For each $i \in [n_r]$, $\mathcal{H}_j$ computes $s_i \leftarrow \alpha_i + \bar{s} \cdot v_i \mod p$ (i.e., $s_i = (\alpha_i + r \cdot v_i + \hat{r} \cdot v_i) + c \cdot gsk \cdot v_i \mod p$) and $s'_i \leftarrow \beta_i + c \cdot v_i \mod p$, where $(c, \bar{s})$ is computed by $\mathcal{H}_j$ as in Figure 3.
6. $\mathcal{H}_j$ sets $\pi_r \leftarrow (\{s_i, s'_i\}_{i=1}^{n_r})$, and outputs a signature $\sigma \leftarrow (T_1, T_2, Y', B, K, \{V_i\}_{i=1}^{n_r}, \pi_2, \pi_r, \mathsf{str})$, where $\mathsf{str} = \mathsf{bsn}$ is unnecessary to be included in the signature if $\mathsf{bsn} \neq \perp$.

For the support of signature-based revocation, a verifier $\mathcal{V}$ is given a signature revocation list `SRL` and additionally checks the validity of $\{V_i\}_{i=1}^{n_r}$ and proof π_r. The changes of the verification algorithm are described as follows:

1. Ignore the check of B and compute $B \leftarrow e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{str}))$ where str is taken from the signature if $\mathsf{bsn} = \perp$ and $\mathsf{str} = \mathsf{bsn}$ otherwise.
2. For every $i \in [n_r]$, check that $V_i \neq 1_{\mathbb{G}_T}$.
3. For each $(\mathsf{str}_i, K_i) \in \mathtt{SRL}$, compute the following commitments $W'_i \leftarrow e(\bar{g}^{s_i}, H_{\mathbb{G}_2}(\mathsf{str}_i)) \cdot K_i^{-s'_i} \cdot V_i^{-c}$ and $Z'_i \leftarrow e(\bar{g}^{s_i}, H_{\mathbb{G}_2}(\mathsf{str})) \cdot K^{-s'_i}$.
4. Compute $c'_h \leftarrow H_2(\text{"sign"}, \bar{g}, g_1, \{h_i\}_{i=0}^n, T_1, T_2, Y', B, K, R_1, R_2, L, \mathtt{SRL}, \{V_i, W'_i, Z'_i\}_{i=1}^{n_r})$ and $c' \leftarrow H_1(N_t, m, \mathsf{str}, D, I, c'_h)$.

E Our DAA Scheme with the Privacy Extension against Subverted TPMs

We extend our DAA scheme $\mathtt{DAA_{OPT}}$ to guarantee privacy against subverted TPMs. Our DAA scheme with subverted TPMs keeps the TPM signing efficiency fully optimal, and outperforms the existing DAA schemes with subverted TPMs [CDL17,CCD+17] in terms of signing performance. Recently, Camenisch et al. [CCD+17] modified the TPM 2.0 commands with minimal changes and used them to implement two ECDAA schemes with subverted TPMs. We can use their modified TPM 2.0 commands to implement our DAA scheme with subverted TPMs.

Following the techniques in [CCD+17], we can extend $\mathtt{DAA_{OPT}}$ to guarantee privacy in the presence of subverted TPMs, and thus avoid a subliminal channel that may be created by a subverted TPM. The extended DAA scheme with subverted TPMs is the same as $\mathtt{DAA_{OPT}}$, except that the join and sign protocols are changed as follows:

- For `TPM.Commit` request, the TPM $\mathcal{M}_i$ picks $N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$ and computes $\bar{N}_t \leftarrow \mathsf{H}(\text{"nonce"}, N_t)$ using a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$ modeled as a random oracle, and then outputs $(E, \bar{N}_t)$ where $E = \bar{g}^r$.
- The host $\mathcal{H}_j$ chooses $N_h \xleftarrow{\$} \{0,1\}^{\ell_n}$, and does the following:
  - In the join protocol, send $\mathtt{TPM.Sign}, (c_h, N_h)$ to $\mathcal{M}_i$.
  - In the sign protocol, send $\mathtt{TPM.Sign}, (m, \mathsf{bsn}, D, I, c_h, N_h)$ to $\mathcal{M}_i$.
- On input $\mathtt{TPM.Sign}, (msg, N_h)$, $\mathcal{M}_i$ computes $c \leftarrow H_1(N_t \oplus N_h, msg)$, and outputs (N_t, s), where msg is either c_h or $(m, \mathsf{bsn}, D, I, c_h)$ and $s = r + c \cdot tsk \mod p$.
- $\mathcal{H}_j$ checks whether $\bar{N}_t = \mathsf{H}(\text{"nonce"}, N_t)$ or not. If the check passes, $\mathcal{H}_j$ computes $N \leftarrow N_t \oplus N_h$, and then re-computes $c \leftarrow H_1(N, msg)$ where msg is defined as in previous step. $\mathcal{H}_j$ checks whether $\bar{g}^s = E \cdot tpk^c$ or not. If the equality holds, $\mathcal{H}_j$ sends (tpk, C, π_t, π_h) to the issuer in the join protocol; or completes the computation of a signature σ and puts a nonce N instead of N_t to σ in the sign protocol.

Since the TPM commits to a nonce N_t before seeing the nonce N_h, and N_t is randomized as $N = N_t \oplus N_h$ by the host, the subverted TPM cannot embed any information into the nonce N. In the random oracle model, $c = H_1(N, msg)$ will be a random value, which cannot be controlled by the subverted TPM. A platform secret key gsk is split into a TPM secret key tsk and a host secret key hsk in a modular addition manner. The host proves knowledge of hsk, which randomizes the E and s from the TPM. Furthermore, the validity of (E, c, s) is also verified by the host. As a result, a subverted TPM cannot embed any information into a signature, and our DAA scheme guarantees privacy against subverted TPMs.

F Functionalities and Simulators

Setup

1. On input (SETUP, sid) from issuer $\mathcal{I}$.
 – Output $(\mathsf{FORWARD}, (\mathsf{SETUP}, sid), \mathcal{I})$ to $\mathcal{S}$.

Join

2. On input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ from host $\mathcal{H}_j$.
 – Output $(\mathsf{FORWARD}, (\mathsf{JOIN}, sid, jsid, \mathcal{M}_i), \mathcal{H}_j)$ to $\mathcal{S}$.
3. On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$ with $attrs \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$.
 – Output $(\mathsf{FORWARD}, (\mathsf{JOINPROCEED}, sid, jsid, attrs), \mathcal{I})$ to $\mathcal{S}$.

Sign

4. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ from host $\mathcal{H}_j$ with $\hat{p} \in \mathbb{P}$.
 – Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}), \mathcal{H}_j)$ to $\mathcal{S}$.
5. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
 – Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i)$ to $\mathcal{S}$.

Verify

6. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
 – Output $(\mathsf{FORWARD}, (\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL}), \mathcal{V})$ to $\mathcal{S}$.

Link

7. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
 – Output $(\mathsf{FORWARD}, (\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}), \mathcal{V})$ to $\mathcal{S}$.

Output

8. On input $(\mathsf{OUTPUT}, \mathcal{P}, m)$ from $\mathcal{S}$.
 – Output (m) to $\mathcal{P}$.

Fig. 6. Functionality $\mathcal{F}$ for Game 3

Setup

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{SETUP}, sid), \mathcal{I})$ from $\mathcal{F}$, $\mathcal{S}$ provides $\mathcal{I}$ with input (SETUP, sid).

Join

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{JOIN}, sid, jsid, \mathcal{M}_i), \mathcal{H}_j)$ from $\mathcal{F}$, $\mathcal{S}$ provides $\mathcal{H}_j$ with input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$.
– Upon receiving $(\mathsf{FORWARD}, (\mathsf{JOINPROCEED}, sid, jsid, attrs), \mathcal{I})$ from $\mathcal{F}$, $\mathcal{S}$ provides "$\mathcal{I}$" with input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$.

Sign

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}), \mathcal{H}_j)$ from $\mathcal{F}$, $\mathcal{S}$ provides "$\mathcal{H}_j$" with input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$.
– Upon receiving $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i)$ from $\mathcal{F}$, $\mathcal{S}$ provides "$\mathcal{M}_i$" with input $(\mathsf{SIGNPROCEED}, sid, ssid)$.

Verify

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL}), \mathcal{V})$ from $\mathcal{F}$, $\mathcal{S}$ provides "$\mathcal{V}$" with input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$.

Link

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}), \mathcal{V})$ from $\mathcal{F}$, $\mathcal{S}$ provides "$\mathcal{V}$" with input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$.

Output

– When any party "$\mathcal{P}$" simulated by $\mathcal{S}$ outputs a message m, $\mathcal{S}$ sends $(\mathsf{OUTPUT}, \mathcal{P}, m)$ to functionality $\mathcal{F}$.

Fig. 7. Simulator for Game 3

Setup

1. `Issuer Setup`. On input (SETUP, sid) from issuer $\mathcal{I}$.
   – Verify that $sid = (\mathcal{I}, sid')$ and output (SETUP, sid) to $\mathcal{S}$.
2. `Set Algorithms`. On input $(\mathsf{ALG}, sid, \mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify})$ from $\mathcal{S}$.
   – Check that ver, link and $\mathsf{identify}$ are deterministic.
   – Store $(sid, \mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

3. On input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ from host $\mathcal{H}_j$.
   – Output $(\mathsf{FORWARD}, (\mathsf{JOIN}, sid, jsid, \mathcal{M}_i), \mathcal{H}_j)$ to $\mathcal{S}$.
4. On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$ with $attrs \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$.
   – Output $(\mathsf{FORWARD}, (\mathsf{JOINPROCEED}, sid, jsid, attrs), \mathcal{I})$ to $\mathcal{S}$.

Sign

5. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ from host $\mathcal{H}_j$ with $\hat{p} \in \mathbb{P}$.
   – Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}), \mathcal{H}_j)$ to $\mathcal{S}$.
6. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   – Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i)$ to $\mathcal{S}$.

Verify

7. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
   – Output $(\mathsf{FORWARD}, (\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL}), \mathcal{V})$ to $\mathcal{S}$.

Link

8. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
   – Output $(\mathsf{FORWARD}, (\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}), \mathcal{V})$ to $\mathcal{S}$.

Output

9. On input $(\mathsf{OUTPUT}, \mathcal{P}, m)$ from $\mathcal{S}$.
   – Output (m) to $\mathcal{P}$.

Fig. 8. Functionality $\mathcal{F}$ for Game 4

Setup

Honest Issuer $\mathcal{I}$

– On input (SETUP, sid) from $\mathcal{F}$.
  - Try to parse sid as $(\mathcal{I}, sid')$ and output $\perp$ to $\mathcal{I}$ if that fails.
  - Provide "$\mathcal{I}$" with input (SETUP, sid).
  - Upon receiving an output $(\mathsf{SETUPDONE}, sid)$ from "$\mathcal{I}$", $\mathcal{S}$ creates its public key $\mathsf{ipk} = (\{h_i\}_{i=0}^{n}, w, \pi_1)$ and secret key $\mathsf{isk} = \gamma$ following the specification of $\mathtt{DAA_{OPT}}$.
  - Define $\mathsf{ukgen}()$ as follows: choose $gsk \leftarrow \mathbb{Z}_p$ and output gsk.
  - Define $\mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ as follows:
    1) Create a BBS+ credential (A, x, u) on gsk and attributes $attrs = (a_1, \ldots, a_n)$ where the disclosed attributes are taken from predicate $\hat{p} = (D, I)$ and the undisclosed attributes are set as dummy values.
    2) Compute $gpk \leftarrow \bar{g}^{gsk}$ and $Y \leftarrow g_1 \cdot gpk \cdot h_0^u \cdot \prod_{i=1}^{n} h_i^{a_i}$.
    3) Following the computational operations at the host side in the real-world sign protocol, randomize A, Y and compute B, K as follows:
       (a) Choose $t_1 \xleftarrow{\$} \mathbb{Z}_p^*$ and $t_2 \xleftarrow{\$} \mathbb{Z}_p$, and compute $T_1 \leftarrow A^{t_1}, T_2 \leftarrow Y^{t_1} \cdot T_1^{-x}$ and $Y' \leftarrow Y^{t_1} \cdot h_0^{-t_2}$.
       (b) If $\mathsf{bsn} = \perp$, pick $b \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $B \leftarrow \bar{g}^b, K \leftarrow gpk^b$; Otherwise, set $B \leftarrow \perp$ and compute $K \leftarrow e(gpk, H_{\mathbb{G}_2}(\mathsf{bsn}))$.
    4) Compute $t_3 = t_1^{-1} \bmod p$ and $\tilde{u} = u - t_2 \cdot t_3 \bmod p$. Without the necessity of distributing the computations between the TPM and host, straightforwardly generate a proof $\pi_2 \leftarrow \mathsf{SPK}_2\big\{(gsk, \{a_i\}_{i\in\bar{D}}, x, \tilde{u}, t_2, t_3) : g_1^{-1} \prod_{i\in D} h_i^{-a_i} = Y'^{-t_3} \bar{g}^{gsk} h_0^{\tilde{u}} \prod_{i\in\bar{D}} h_i^{a_i} \wedge\ T_2/Y' = T_1^{-x} h_0^{t_2}\ \wedge K = B^{gsk}\big\}(\text{"sign"}, m, \mathsf{bsn}, D, I)$ as below:
       (a) Choose $\bar{r}, r_x, r_{\tilde{u}}, r_{t_2}, r_{t_3} \xleftarrow{\$} \mathbb{Z}_p$, $N_t \xleftarrow{\$} \{0,1\}^{\ell_n}$, and $r_{a_i} \xleftarrow{\$} \mathbb{Z}_p$ for each $i \in \bar{D}$.
       (b) Compute $R_1 \leftarrow Y'^{-r_{t_3}} \cdot \bar{g}^{\bar{r}} \cdot h_0^{r_{\tilde{u}}} \cdot \prod_{i\in\bar{D}} h_i^{r_{a_i}}$, $R_2 \leftarrow T_1^{-r_x} \cdot h_0^{r_{t_2}}$
       (c) Calculate $L \leftarrow B^{\bar{r}}$ if $\mathsf{bsn} = \perp$ and $L \leftarrow e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))^{\bar{r}}$ otherwise.
       (d) Compute $c_h \leftarrow H_2(\text{"sign"}, \bar{g}, g_1, \{h_i\}_{i=0}^{n}, T_1, T_2, Y', B, K, R_1, R_2, L)$.
       (e) Compute $c \leftarrow H_1(N_t, m, \mathsf{bsn}, D, I, c_h)$.
       (f) Compute $\bar{s} \leftarrow \bar{r} + c \cdot gsk \bmod p$, $s_x \leftarrow r_x + c \cdot x \bmod p$, $s_{\tilde{u}} \leftarrow r_{\tilde{u}} + c \cdot \tilde{u} \bmod p$, $s_{t_2} \leftarrow r_{t_2} + c \cdot t_2 \bmod p$, $s_{t_3} \leftarrow r_{t_3} + c \cdot t_3 \bmod p$, and $s_{a_i} \leftarrow r_{a_i} + c \cdot a_i \bmod p$ for each $i \in \bar{D}$ where a_i is a dummy value.
       (g) Set $\pi_2 \leftarrow (c, \bar{s}, s_x, s_{\tilde{u}}, s_{t_2}, s_{t_3}, \{s_{a_i}\}_{i\in\bar{D}}, N_t)$.
    5) Output a signature $\sigma \leftarrow (T_1, T_2, Y', B, K, \pi_2)$.
  - Define $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$ as the real world verification algorithm except that the private key revocation check is omitted.
  - Define $\mathsf{link}(m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ as follows: 1) parse the signatures σ_0 and σ_1 as $(T_{1,0}, T_{2,0}, Y_0', B_0, K_0, \pi_{2,0})$ and $(T_{1,1}, T_{2,1}, Y_1', B_1, K_1, \pi_{2,1})$ respectively; 2) output 1 if $K_0 = K_1$ and 0 otherwise.
  - Define $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk)$ as follows: 1) parse σ as $(T_1, T_2, Y', B, K, \pi_2)$; 2) compute $B \leftarrow e(\bar{g}, H_{\mathbb{G}_2}(\mathsf{bsn}))$ if $\mathsf{bsn} \neq \perp$; 3) check $gsk \in \mathbb{Z}_p$ and $K = B^{gsk}$; 4) output 1 if the check passes and 0 otherwise.
  - $\mathcal{S}$ sends $(\mathsf{ALG}, sid, \mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify})$ to $\mathcal{F}$.

Corrupt Issuer $\mathcal{I}$

– $\mathcal{S}$ notices this setup as it notices $\mathcal{I}$ registering a public key with "$\mathcal{F}_{\mathsf{ca}}$" with $sid = (\mathcal{I}, sid')$.
  - If the registered key ipk is of the form $h_0, h_1, \ldots, h_n, w, \pi_1$ and the proof π_1 is valid, $\mathcal{S}$ uses Ext_1 to extract a secret key γ from proof π_1.
  - $\mathcal{S}$ defines the algorithms $\mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}$ as before, but now relying on the extracted secret key.
  - $\mathcal{S}$ sends (SETUP, sid) to $\mathcal{F}$ on behalf of $\mathcal{I}$.

– $\mathcal{S}$ sends $(\mathsf{ALG}, sid, \mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify})$ to $\mathcal{F}$.

Join, Sign, Verify, Link

– Unchanged.

Output

– When any simulated party "$\mathcal{P}$" outputs a message m which is not explicitly handled by $\mathcal{S}$ yet, $\mathcal{S}$ sends $(\mathsf{OUTPUT}, \mathcal{P}, m)$ to $\mathcal{F}$.

Fig. 9. Simulator for Game 4

Setup

Unchanged

Join

3. On input ($\mathsf{JOIN}, sid, jsid, \mathcal{M}_i$) from host $\mathcal{H}_j$.
   – Output ($\mathsf{FORWARD}, (\mathsf{JOIN}, sid, jsid, \mathcal{M}_i), \mathcal{H}_j$) to $\mathcal{S}$.
4. On input ($\mathsf{JOINPROCEED}, sid, jsid, attrs$) from $\mathcal{I}$ with $attrs \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$.
   – Output ($\mathsf{FORWARD}, (\mathsf{JOINPROCEED}, sid, jsid, attrs), \mathcal{I}$) to $\mathcal{S}$.

Sign

5. On input ($\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}$) from host $\mathcal{H}_j$ with $\hat{p} \in \mathbb{P}$.
   – Output ($\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}), \mathcal{H}_j$) to $\mathcal{S}$.
6. On input ($\mathsf{SIGNPROCEED}, sid, ssid$) from $\mathcal{M}_i$.
   – Output ($\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i$) to $\mathcal{S}$.

Verify

7. `Verify`. On input ($\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL}$) from some party $\mathcal{V}$.
   – Set $f \leftarrow 0$ if at least one of the following conditions hold:
     • There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   – If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   – Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output ($\mathsf{VERIFIED}, sid, f$) to $\mathcal{V}$.

Link

8. `Link`. On input ($\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}$) from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
   – Output $\perp$ to $\mathcal{V}$ if at least one signature tuple ($m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0$) or ($m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1$) is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
   – Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
   – Output (LINK, sid, f) to $\mathcal{V}$.

Output

9. On input ($\mathsf{OUTPUT}, \mathcal{P}, m$) from $\mathcal{S}$.
   – Output (m) to $\mathcal{P}$.

Fig. 10. Functionality $\mathcal{F}$ for Game 5

Setup

– Unchanged.

Join

– Unchanged.

Sign

– Unchanged.

Verify

– Nothing to simulate.

Link

– Nothing to simulate.

Output

– When any simulated party "$\mathcal{P}$" outputs a message m which is not explicitly handled by $\mathcal{S}$ yet, $\mathcal{S}$ sends ($\mathsf{OUTPUT}, \mathcal{P}, m$) to $\mathcal{F}$.

Fig. 11. Simulator for Game 5

Setup

Unchanged

Join

3. `Join Request`. On input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ from host $\mathcal{H}_j$.
   - Create a join session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
4. `Join Request Delivery`. On input $(\mathsf{JOINSTART}, sid, jsid)$ from $\mathcal{S}$.
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $status \leftarrow delivered$.
   - Abort if $\mathcal{I}$ or $\mathcal{M}_i$ is honest and a record $\langle \mathcal{M}_i, *, *, *\rangle \in$ `Members` already exists.
   - Output $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ to $\mathcal{I}$.
5. `Join Proceed`. On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$ with $attrs \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$.
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $\perp \leftarrow attrs$ and $status \leftarrow complete$.
   - Output $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ to $\mathcal{S}$.
6. `Platform Key Generation`. On input $(\mathsf{JOINCOMPLETE}, sid, jsid, gsk)$ from $\mathcal{S}$.
   - Look up record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ with $status = complete$.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, set $gsk \leftarrow \perp$.
   - Add $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Memebers` and output $(\mathsf{JOINED}, sid, jsid)$ to $\mathcal{H}_j$.

Sign

7. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ from host $\mathcal{H}_j$ with $\hat{p} \in \mathbb{P}$.
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}), \mathcal{H}_j)$ to $\mathcal{S}$.
8. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i)$ to $\mathcal{S}$.

Verify

9. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
   - Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f\rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

10. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
    - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
    - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
    - Output (LINK, sid, f) to $\mathcal{V}$.

Output

11. On input $(\mathsf{OUTPUT}, \mathcal{P}, m)$ from $\mathcal{S}$.
    - Output (m) to $\mathcal{P}$.

Fig. 12. Functionality $\mathcal{F}$ for Game 6

Setup, Sign: Unchanged.

Join

Honest $\mathcal{H}, \mathcal{I}$

- $\mathcal{S}$ receives $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ simulates the real-world protocol via giving "$\mathcal{H}_j$" input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ and waits for output $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ from "$\mathcal{I}$".
  - If $\mathcal{M}_i$ is honest, $\mathcal{S}$ knows tsk as it is simulating $\mathcal{M}_i$. If $\mathcal{M}_i$ is corrupted, $\mathcal{S}$ runs Ext_t to extract tsk from proof π_t. Since $\mathcal{S}$ simulates the honest host $\mathcal{H}_j$, it knows hsk. Finally, $\mathcal{S}$ computes $gsk \leftarrow tsk + hsk \mod p$.
  - $\mathcal{S}$ sends $(\mathsf{JOINSTART}, sid, jsid)$ to $\mathcal{F}$.
- On input $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation by giving "$\mathcal{I}$" input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$, and waits for output $(\mathsf{JOINED}, sid, jsid)$ from "$\mathcal{H}_j$".
  - Output $(\mathsf{JOINCOMPLETE}, sid, jsid, gsk)$ to $\mathcal{F}$.

Honest $\mathcal{H}$, Corrupt $\mathcal{I}$

- On input $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ simulates the real-world protocol via giving "$\mathcal{H}_j$" input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ and waits for output $(\mathsf{JOINED}, sid, jsid)$ from "$\mathcal{H}_j$". $\mathcal{S}$ knows which attributes $attrs$ the corrupted issuer issued to "$\mathcal{H}_j$", as it simulates "$\mathcal{H}_j$".
  - $\mathcal{S}$ sends $(\mathsf{JOINSTART}, sid, jsid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ from $\mathcal{F}$, $\mathcal{S}$ sends $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ to $\mathcal{F}$ on behalf of $\mathcal{I}$.
- Upon receiving $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ from $\mathcal{F}$, $\mathcal{S}$ sends $(\mathsf{JOINCOMPLETE}, sid, jsid, \perp)$ to $\mathcal{F}$.

Honest $\mathcal{M}, \mathcal{I}$, Corrupt $\mathcal{H}$

- $\mathcal{S}$ notices the join as "$\mathcal{I}$" outputs $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$.
  - $\mathcal{S}$ knows the identity of the host $\mathcal{H}_j$ involved in the join session, as it is simulating "$\mathcal{M}_i$".
  - $\mathcal{S}$ takes tsk from simulating "$\mathcal{M}_i$", and runs Ext_h to extract hsk from proof π_h. Then $\mathcal{S}$ sets $gsk \leftarrow tsk + hsk \mod p$.
  - $\mathcal{S}$ sends $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ on behalf of $\mathcal{H}_j$ to $\mathcal{F}$.
- $\mathcal{S}$ receives $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation of "$\mathcal{M}_i$" until "$\mathcal{I}$" outputs $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$.
  - $\mathcal{S}$ sends $(\mathsf{JOINSTART}, sid, jsid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ from $\mathcal{F}$, $\mathcal{S}$ sends $(\mathsf{JOINCOMPLETE}, sid, jsid, gsk)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{JOINED}, sid, jsid)$ from $\mathcal{F}$ as host $\mathcal{H}_j$ is corrupted, $\mathcal{S}$ continues the simulation via giving "$\mathcal{I}$" input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$.

Honest $\mathcal{I}$, Corrupt $\mathcal{M}, \mathcal{H}$

- $\mathcal{S}$ notices the join as "$\mathcal{I}$" receives $(\mathsf{SENT}, (\mathcal{M}_i, \mathcal{I}, sid'), jsid, (tpk, C, \pi_t, \pi_h), \mathcal{H}'_j)$ from $\mathcal{F}_{\mathsf{auth}*}$.
  - $\mathcal{S}$ runs Ext_t to extract tsk from proof π_t and uses Ext_h to extract hsk from proof π_h. Then, $\mathcal{S}$ sets $gsk \leftarrow tsk + hsk \mod p$.
  - $\mathcal{S}$ does not know the exact identity of the host who launched the join session. So, $\mathcal{S}$ chooses an arbitrary corrupt host $\mathcal{H}_j$ and proceeds as if it is the host who launched the join session. For corrupt platforms, the exact identity of the host does not matter.
  - $\mathcal{S}$ sends $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ to $\mathcal{F}$ on behalf of $\mathcal{H}_j$.
- $\mathcal{S}$ receives $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues simulating "$\mathcal{I}$" until it outputs $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$.
  - $\mathcal{S}$ sends $(\mathsf{JOINSTART}, sid, jsid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ from $\mathcal{F}$, $\mathcal{S}$ sends $(\mathsf{JOINCOMPLETE}, sid, jsid, gsk)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{JOINED}, sid, jsid)$ from $\mathcal{F}$ as host $\mathcal{H}_j$ is corrupted, $\mathcal{S}$ continues the simulation by giving "$\mathcal{I}$" input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$.

Honest $\mathcal{M}$, Corrupt $\mathcal{H}, \mathcal{I}$

- $\mathcal{S}$ notices this join as "$\mathcal{M}_i$" receives a message $(\mathtt{TPM.Create}, sid, jsid)$ from host $\mathcal{H}_j$.
- $\mathcal{S}$ simply simulates "$\mathcal{M}_i$" honestly, and does not need to involve $\mathcal{F}$, since $\mathcal{M}_i$ does not receive inputs or send outputs in the join related interfaces, and $\mathcal{F}$ does not guarantee any security property for platforms with corrupt hosts when the issuer is corrupted.

Verify, Link: Nothing to simulate.

Output: When any simulated party "$\mathcal{P}$" outputs a message m which is not explicitly handled by $\mathcal{S}$ yet, $\mathcal{S}$ sends $(\mathsf{OUTPUT}, \mathcal{P}, m)$ to $\mathcal{F}$.

Fig. 13. Simulator for Game 6

Setup

Unchanged

Join

Unchanged

Sign with $\mathsf{bsn} \neq \perp$

7. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ from host $\mathcal{H}_j$ with $\hat{p} \in \mathbb{P}$.
 – Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}), \mathcal{H}_j)$ to $\mathcal{S}$.
8. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
 – Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i)$ to $\mathcal{S}$.

Sign with $\mathsf{bsn} = \perp$

9. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \perp$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
 – Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status \leftarrow request$.
 – Output $(\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
10. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
 – Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ to $status \leftarrow delivered$.
 – Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
11. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
 – Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status = delivered$.
 – Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
12. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
 – If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
  • As $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$, and then store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle$ in `DomainKeys`.
  • Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$.
 – If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p}\rangle$ in `Signed`.
 – Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
 – Set $f \leftarrow 0$ if at least one of the following conditions hold:
  • There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
 – If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
 – Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f\rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
 – Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
 – Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
 – Output (LINK, sid, f) to $\mathcal{V}$.

Output

15. On input $(\mathsf{OUTPUT}, \mathcal{P}, m)$ from $\mathcal{S}$.
 – Output (m) to $\mathcal{P}$.

Fig. 14. Functionality $\mathcal{F}$ for Game 7

Setup

– Unchanged.

Join

– Unchanged.

Sign with $\mathsf{bsn} \neq \perp$

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}), \mathcal{H}_j)$ from $\mathcal{F}$, $\mathcal{S}$ provides "$\mathcal{H}_j$" with input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$.

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i)$ from $\mathcal{F}$, $\mathcal{S}$ provides "$\mathcal{M}_i$" with input $(\mathsf{SIGNPROCEED}, sid, ssid)$.

Sign with $\mathsf{bsn} = \perp$

Honest $\mathcal{M}, \mathcal{H}$

– Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j)$ with $\mathsf{bsn} = \perp$ from $\mathcal{F}$.
  - $\mathcal{S}$ starts the simulation via giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$, $\mathcal{S}$ sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, $\mathcal{S}$ sends $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.

Honest $\mathcal{H}$, Corrupt $\mathcal{M}$

– Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j)$ with $\mathsf{bsn} = \perp$ from $\mathcal{F}$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ from $\mathcal{F}$ as $\mathcal{M}_i$ is corrupted.
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, $\mathcal{S}$ sends $(\mathsf{SIGNPROCEED}, sid, ssid)$ to $\mathcal{F}$ on behalf of $\mathcal{M}_i$.

– Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ to $\mathcal{F}$.

Honest $\mathcal{M}$, Corrupt $\mathcal{H}$

– $\mathcal{S}$ notices this sign session as "$\mathcal{M}_i$" receives $(sid, ssid, m, \mathsf{bsn}, \hat{p})$ from $\mathcal{H}_j$.
  - $\mathcal{S}$ sends $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{F}$ on behalf of $\mathcal{H}_j$.

– Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation of "$\mathcal{M}_i$" until it outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNATURE}, sid, ssid, \perp)$ from $\mathcal{F}$ as $\mathcal{H}_j$ is corrupted.
  - $\mathcal{S}$ continues the simulation via giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.

Verify

– Nothing to simulate.

Link

– Nothing to simulate.

Output

– When any simulated party "$\mathcal{P}$" outputs a message m which is not explicitly handled by $\mathcal{S}$ yet, $\mathcal{S}$ sends $(\mathsf{OUTPUT}, \mathcal{P}, m)$ to $\mathcal{F}$.

Fig. 15. Simulator for Game 7

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \bot$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
     - If $\mathsf{bsn} \neq \bot$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \bot$, generate $gsk \leftarrow \mathsf{ukgen}()$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle$ in `DomainKeys`.
     - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$.
   - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p}\rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
   - Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f\rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \bot$.
   - Output $\bot$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
   - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 16. Functionality $\mathcal{F}$ for Game 8

Setup

– Unchanged.

Join

– Unchanged.

Sign

Honest $\mathcal{M}, \mathcal{H}$

– Upon receiving ($\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j$) from $\mathcal{F}$.
  - $\mathcal{S}$ starts the simulation via giving "$\mathcal{H}_j$" input ($\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}$).
  - When "$\mathcal{M}_i$" outputs ($\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p}$), $\mathcal{S}$ sends ($\mathsf{SIGNSTART}, sid, ssid$) to $\mathcal{F}$.

– Upon receiving ($\mathsf{SIGNCOMPLETE}, sid, ssid$) from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation by giving "$\mathcal{M}_i$" input ($\mathsf{SIGNPROCEED}, sid, ssid$).
  - When "$\mathcal{H}_j$" outputs ($\mathsf{SIGNATURE}, sid, ssid, \sigma$), $\mathcal{S}$ sends ($\mathsf{SIGNCOMPLETE}, sid, ssid, \perp$) to $\mathcal{F}$.

Honest $\mathcal{H}$, *Corrupt* $\mathcal{M}$

– Upon receiving ($\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j$) from $\mathcal{F}$.
  - $\mathcal{S}$ sends ($\mathsf{SIGNSTART}, sid, ssid$) to $\mathcal{F}$.

– Upon receiving ($\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p}$) from $\mathcal{F}$ as $\mathcal{M}_i$ is corrupted.
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{H}_j$" input ($\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}$).
  - When "$\mathcal{H}_j$" outputs ($\mathsf{SIGNATURE}, sid, ssid, \sigma$), $\mathcal{S}$ sends ($\mathsf{SIGNPROCEED}, sid, ssid$) to $\mathcal{F}$ on behalf of $\mathcal{M}_i$.

– Upon receiving ($\mathsf{SIGNCOMPLETE}, sid, ssid$) from $\mathcal{F}$.
  - $\mathcal{S}$ sends ($\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma$) to $\mathcal{F}$.

Honest $\mathcal{M}$, *Corrupt* $\mathcal{H}$

– $\mathcal{S}$ notices this sign session as "$\mathcal{M}_i$" receives ($sid, ssid, m, \mathsf{bsn}, \hat{p}$) from $\mathcal{H}_j$.
  - $\mathcal{S}$ sends ($\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}$) to $\mathcal{F}$ on behalf of $\mathcal{H}_j$.

– Upon receiving ($\mathsf{SIGNSTART}, sid, ssid, m, \mathsf{bsn}, \hat{p}, \mathcal{M}_i, \mathcal{H}_j$) from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation of "$\mathcal{M}_i$" until it outputs ($\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p}$).
  - $\mathcal{S}$ sends ($\mathsf{SIGNSTART}, sid, ssid$) to $\mathcal{F}$.

– Upon receiving ($\mathsf{SIGNCOMPLETE}, sid, ssid$) from $\mathcal{F}$.
  - $\mathcal{S}$ sends ($\mathsf{SIGNCOMPLETE}, sid, ssid, \perp$) to $\mathcal{F}$.

– Upon receiving ($\mathsf{SIGNATURE}, sid, ssid, \perp$) from $\mathcal{F}$ as $\mathcal{H}_j$ is corrupted.
  - $\mathcal{S}$ continues the simulation via giving "$\mathcal{M}_i$" input ($\mathsf{SIGNPROCEED}, sid, ssid$).

Verify

– Nothing to simulate.

Link

– Nothing to simulate.

Fig. 17. Simulator for Game 8

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input ($\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}$) with $\mathsf{bsn} = \perp$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status \leftarrow request$.
   - Output ($\mathsf{SIGNSTART}, sid, ssid, \boldsymbol{l}(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j$) to $\mathcal{S}$.
8. `Sign Request Delivery`. On input ($\mathsf{SIGNSTART}, sid, ssid$) from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ to $status \leftarrow delivered$.
   - Output ($\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p}$) to $\mathcal{M}_i$.
9. `Sign Proceed`. On input ($\mathsf{SIGNPROCEED}, sid, ssid$) from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status = delivered$.
   - Output ($\mathsf{SIGNCOMPLETE}, sid, ssid$) to $\mathcal{S}$.
10. `Signature Generation`. On input ($\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma$) from $\mathcal{S}$.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
     - If $\mathsf{bsn} \neq \perp$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle$ in `DomainKeys`.
     - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$.
   - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p} \rangle$ in `Signed`.
   - Output ($\mathsf{SIGNATURE}, sid, ssid, \sigma$) to $\mathcal{H}_j$.

Verify

13. `Verify`. On input ($\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL}$) from some party $\mathcal{V}$.
   - Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output ($\mathsf{VERIFIED}, sid, f$) to $\mathcal{V}$.

Link

14. `Link`. On input ($\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}$) from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
   - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 18. Functionality $\mathcal{F}$ for Game 9

Setup

– Unchanged.

Join

– Unchanged.

Sign

Honest $\mathcal{M}, \mathcal{H}$

– Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ takes a dummy message m', basename bsn' and attribute predicate $\hat{p}'$ such that $l(m', \mathsf{bsn}', \hat{p}') = l$ and $\hat{p}'$ holds for the platform's attributes which are learned by $\mathcal{S}$ from the join protocol.
  - $\mathcal{S}$ starts the simulation via giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m', \mathsf{bsn}', \hat{p}')$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m', \mathsf{bsn}', \hat{p}')$, $\mathcal{S}$ sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, $\mathcal{S}$ sends $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.

Honest $\mathcal{H}$, *Corrupt* $\mathcal{M}$

– Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ from $\mathcal{F}$ as $\mathcal{M}_i$ is corrupted.
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, $\mathcal{S}$ sends $(\mathsf{SIGNPROCEED}, sid, ssid)$ to $\mathcal{F}$ on behalf of $\mathcal{M}_i$.

– Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ to $\mathcal{F}$.

Honest $\mathcal{M}$, *Corrupt* $\mathcal{H}$

– $\mathcal{S}$ notices this sign session as "$\mathcal{M}_i$" receives $(sid, ssid, m, \mathsf{bsn}, \hat{p})$ from $\mathcal{H}_j$.
  - $\mathcal{S}$ sends $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{F}$ on behalf of $\mathcal{H}_j$.

– Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation of "$\mathcal{M}_i$" until it outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ sends $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNATURE}, sid, ssid, \perp)$ from $\mathcal{F}$ as $\mathcal{H}_j$ is corrupted.
  - $\mathcal{S}$ continues the simulation via giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.

Verify

– Nothing to simulate.

Link

– Nothing to simulate.

Fig. 19. Simulator for Game 9

Setup

Unchanged

Join

3. `Join Request`. On input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ from host $\mathcal{H}_j$.
  - Create a join session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, status\rangle$ with $status \leftarrow request$.
  - Output $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
4. `Join Request Delivery`. On input $(\mathsf{JOINSTART}, sid, jsid)$ from $\mathcal{S}$.
  - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $status \leftarrow delivered$.
  - Abort if $\mathcal{I}$ or $\mathcal{M}_i$ is honest and a record $\langle \mathcal{M}_i, *, *, *\rangle \in \mathtt{Members}$ already exists.
  - Output $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ to $\mathcal{I}$.
5. `Join Proceed`. On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$ with $attrs \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$.
  - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $\perp \leftarrow attrs$ and $status \leftarrow complete$.
  - Output $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs')$ to $\mathcal{S}$, where $attrs' \leftarrow \perp$ if $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest and $attrs' \leftarrow attrs$ otherwise.
6. `Platform Key Generation`. On input $(\mathsf{JOINCOMPLETE}, sid, jsid, gsk)$ from $\mathcal{S}$.
  - Look up record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ with $status = complete$.
  - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, set $gsk \leftarrow \perp$.
  - Add $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Memebers` and output $(\mathsf{JOINED}, sid, jsid)$ to $\mathcal{H}_j$.

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \perp$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
  - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
  - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status \leftarrow request$.
  - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
  - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ to $status \leftarrow delivered$.
  - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
  - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status = delivered$.
  - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
  - If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
  - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
    - If $\mathsf{bsn} \neq \perp$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle \in \mathtt{DomainKeys}$ for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle$ in `DomainKeys`.
    - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$.
  - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p}\rangle$ in `Signed`.
  - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
  - Set $f \leftarrow 0$ if at least one of the following conditions hold:
    - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
  - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
  - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f\rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
  - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
  - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
  - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 20. Functionality $\mathcal{F}$ for Game 10

Setup

– Unchanged.

Join

Honest $\mathcal{M}, \mathcal{H}, \mathcal{I}$

– Upon receiving $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$, $\mathcal{S}$ does the following:
  - $\mathcal{S}$ simulates the real-world join protocol via giving "$\mathcal{H}_j$" input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$.
  - When "$\mathcal{I}$" outputs $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$, $\mathcal{S}$ sends $(\mathsf{JOINSTART}, sid, jsid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ from $\mathcal{F}$.
  - $\mathcal{S}$ does not know the attributes, as it receives $attrs = \bot$. Thus, $\mathcal{S}$ picks a random $attrs' \in \mathbb{A}_1 \times \ldots \mathbb{A}_n$.
  - $\mathcal{S}$ continues the simulation by giving "$\mathcal{I}$" input $(\mathsf{JOINPROCEED}, sid, jsid, attrs')$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{JOINED}, sid, jsid)$, $\mathcal{S}$ outputs $(\mathsf{JOINCOMPLETE}, sid, jsid, gsk)$ to $\mathcal{F}$.

Other Cases

– Unchanged.

Sign

Honest $\mathcal{M}, \mathcal{H}$

– Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ takes a dummy message m', basename bsn' and attribute predicate $\hat{p}'$ such that $l(m', \mathsf{bsn}', \hat{p}') = l$ and $\hat{p}'$ holds for the dummy attributes that are chosen at random by $\mathcal{S}$ in the join protocol.
  - $\mathcal{S}$ starts the simulation via giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m', \mathsf{bsn}', \hat{p}')$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m', \mathsf{bsn}', \hat{p}')$, $\mathcal{S}$ sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.

– Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ continues the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, $\mathcal{S}$ sends $(\mathsf{SIGNCOMPLETE}, sid, ssid, \bot)$ to $\mathcal{F}$.

Other Cases

– Unchanged.

Verify

– Nothing to simulate.

Link

– Nothing to simulate.

Fig. 21. Simulator for Game 10

Setup

Unchanged

Join

3. `Join Request`. On input (JOIN, $sid, jsid, \mathcal{M}_i$) from host $\mathcal{H}_j$.
  – Create a join session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, status\rangle$ with $status \leftarrow request$.
  – Output (JOINSTART, $sid, jsid, \mathcal{M}_i, \mathcal{H}_j$) to $\mathcal{S}$.
4. `Join Request Delivery`. On input (JOINSTART, $sid, jsid$) from $\mathcal{S}$.
  – Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $status \leftarrow delivered$.
  – Abort if $\mathcal{I}$ or $\mathcal{M}_i$ is honest and a record $\langle \mathcal{M}_i, *, *, *\rangle \in$ `Members` already exists.
  – Output (JOINPROCEED, $sid, jsid, \mathcal{M}_i$) to $\mathcal{I}$.
5. `Join Proceed`. On input (JOINPROCEED, $sid, jsid, attrs$) from $\mathcal{I}$ with $attrs \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$.
  – Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ to $\perp \leftarrow attrs$ and $status \leftarrow complete$.
  – Output (JOINCOMPLETE, $sid, jsid, attrs'$) to $\mathcal{S}$, where $attrs' \leftarrow \perp$ if $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest and $attrs' \leftarrow attrs$ otherwise.
6. `Platform Key Generation`. On input (JOINCOMPLETE, $sid, jsid, gsk$) from $\mathcal{S}$.
  – Look up record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ with $status = complete$.
  – If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, set $gsk \leftarrow \perp$.
  – Else verify that the provided gsk is eligible via checking
    • $\mathsf{CheckGskHonest}(gsk) = 1$ if $\mathcal{M}_i$ is honest and $\mathcal{H}_j$ is corrupted.
    • $\mathsf{CheckGskCorrupt}(gsk) = 1$ if $\mathcal{M}_i$ is corrupted.
  – Add $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Memebers` and output (JOINED, $sid, jsid$) to $\mathcal{H}_j$.

Sign

7. `Sign Request`. On input (SIGN, $sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}$) with $\mathsf{bsn} = \perp$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
  – If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
  – Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status \leftarrow request$.
  – Output (SIGNSTART, $sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j$) to $\mathcal{S}$.
8. `Sign Request Delivery`. On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$.
  – Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ to $status \leftarrow delivered$.
  – Output (SIGNPROCEED, $sid, ssid, m, \mathsf{bsn}, \hat{p}$) to $\mathcal{M}_i$.
9. `Sign Proceed`. On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$.
  – Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status = delivered$.
  – Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
10. `Signature Generation`. On input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$.
  – If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
  – If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
    • If $\mathsf{bsn} \neq \perp$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle$ in `DomainKeys`.
    • Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$.
  – If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p}\rangle$ in `Signed`.
  – Output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{H}_j$.

Verify

13. `Verify`. On input (VERIFY, $sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL}$) from some party $\mathcal{V}$.
  – Set $f \leftarrow 0$ if at least one of the following conditions hold:
    • There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
  – If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
  – Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f\rangle$ to `VerResults` and output (VERIFIED, sid, f) to $\mathcal{V}$.

Link

14. `Link`. On input (LINK, $sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}$) from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
  – Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
  – Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
  – Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 22. Functionality $\mathcal{F}$ for Game 11

Setup
– Unchanged.
Join
– Unchanged.
Sign
– Unchanged.
Verify
– Unchanged.
Link
– Unchanged.

Fig. 23. Simulators for Games 11-17

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \perp$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
   - If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
     - If $\mathsf{bsn} \neq \perp$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle$ in `DomainKeys`.
     - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ and check $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$.
     - Check that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
   - Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
   - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 24. Functionality $\mathcal{F}$ for Game 12

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \bot$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
    - If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
    - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
      - If $\mathsf{bsn} \neq \bot$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \bot$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle$ in `DomainKeys`.
      - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ and check $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$.
      - Check that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
    - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p} \rangle$ in `Signed`.
    - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
    - Retrieve all pairs $(\mathcal{M}_i, gsk_i)$ from $\langle \mathcal{M}_i, *, gsk_i, * \rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i \rangle \in$ `DomainKeys` such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
      - More than one key gsk_i was found.
      - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
    - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
    - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \bot$.
    - Output $\bot$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
    - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
    - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 25. Functionality $\mathcal{F}$ for Game 13

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \bot$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
   – If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
   – Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status \leftarrow request$.
   – Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   – Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ to $status \leftarrow delivered$.
   – Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   – Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status = delivered$.
   – Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
   – If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
   – If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
      • If $\mathsf{bsn} \neq \bot$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \bot$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle$ in `DomainKeys`.
      • Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ and check $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$.
      • Check that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   – If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p} \rangle$ in `Signed`.
   – Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
   – Retrieve all pairs $(\mathcal{M}_i, gsk_i)$ from $\langle \mathcal{M}_i, *, gsk_i, * \rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i \rangle \in$ `DomainKeys` such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
      • More than one key gsk_i was found.
      • $\mathcal{I}$ is honest and no pair $(\mathcal{M}_i, gsk_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in$ `Members` with $\hat{p}(attrs) = 1$ exists.
      • There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   – If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   – Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \bot$.
   – Output $\bot$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
   – Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
   – Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 26. Functionality $\mathcal{F}$ for Game 14

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \bot$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.
9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
   - If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
     - If $\mathsf{bsn} \neq \bot$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \bot$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle$ in `DomainKeys`.
     - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ and check $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$.
     - Check that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
   - Retrieve all pairs $(\mathcal{M}_i, gsk_i)$ from $\langle \mathcal{M}_i, *, gsk_i, * \rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i \rangle \in$ `DomainKeys` such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - More than one key gsk_i was found.
     - $\mathcal{I}$ is honest and no pair $(\mathcal{M}_i, gsk_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in$ `Members` with $\hat{p}(attrs) = 1$ exists.
     - There is an honest $\mathcal{M}_i$ but no entry $\langle m, \mathsf{bsn}, *, \mathcal{M}_i, \hat{p} \rangle \in$ `Signed` exists.
     - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \bot$.
   - Output $\bot$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
   - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 27. Functionality $\mathcal{F}$ for Game 15

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input ($\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p}$) with $\mathsf{bsn} = \perp$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status \leftarrow request$.
   - Output ($\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j$) to $\mathcal{S}$.
8. `Sign Request Delivery`. On input ($\mathsf{SIGNSTART}, sid, ssid$) from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ to $status \leftarrow delivered$.
   - Output ($\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p}$) to $\mathcal{M}_i$.
9. `Sign Proceed`. On input ($\mathsf{SIGNPROCEED}, sid, ssid$) from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status \rangle$ with $status = delivered$.
   - Output ($\mathsf{SIGNCOMPLETE}, sid, ssid$) to $\mathcal{S}$.
10. `Signature Generation`. On input ($\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma$) from $\mathcal{S}$.
   - If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
     - If $\mathsf{bsn} \neq \perp$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle \in$ `DomainKeys` for ($\mathcal{M}_i, \mathsf{bsn}$). If no such gsk exists or $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk \rangle$ in `DomainKeys`.
     - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ and check $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$.
     - Check that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p} \rangle$ in `Signed`.
   - Output ($\mathsf{SIGNATURE}, sid, ssid, \sigma$) to $\mathcal{H}_j$.

Verify

13. `Verify`. On input ($\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL}$) from some party $\mathcal{V}$.
   - Retrieve all pairs ($\mathcal{M}_i, gsk_i$) from $\langle \mathcal{M}_i, *, gsk_i, * \rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i \rangle \in$ `DomainKeys` such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - More than one key gsk_i was found.
     - $\mathcal{I}$ is honest and no pair ($\mathcal{M}_i, gsk_i$) was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in$ `Members` with $\hat{p}(attrs) = 1$ exists.
     - There is an honest $\mathcal{M}_i$ but no entry $\langle m, \mathsf{bsn}, *, \mathcal{M}_i, \hat{p} \rangle \in$ `Signed` exists.
     - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$ **and no pair ($\boldsymbol{\mathcal{M}_i, gsk_i}$) for an honest $\boldsymbol{\mathcal{M}_i}$ was found.**
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
   - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f \rangle$ to `VerResults` and output ($\mathsf{VERIFIED}, sid, f$) to $\mathcal{V}$.

Link

14. `Link`. On input ($\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn}$) from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple ($m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0$) or ($m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1$) is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
   - Set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 28. Functionality $\mathcal{F}$ for Game 16

Setup

Unchanged

Join

Unchanged

Sign

7. `Sign Request`. On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, \mathsf{bsn}, \hat{p})$ with $\mathsf{bsn} = \perp$ and $\hat{p} \in \mathbb{P}$ from host $\mathcal{H}_j$.
  - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`, abort.
  - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status \leftarrow request$.
  - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, \mathsf{bsn}, \hat{p}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.

8. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
  - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ to $status \leftarrow delivered$.
  - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, \mathsf{bsn}, \hat{p})$ to $\mathcal{M}_i$.

9. `Sign Proceed`. On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
  - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, \mathsf{bsn}, \hat{p}, status\rangle$ with $status = delivered$.
  - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.

10. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
  - If $\mathcal{I}$ is honest, check that $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $\hat{p}(attrs) = 1$ exists in `Members`.
  - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the signature σ from $\mathcal{S}$ and internally generate a signature for a fresh or established gsk:
    - If $\mathsf{bsn} \neq \perp$, retrieve gsk from $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such gsk exists or $\mathsf{bsn} = \perp$, generate $gsk \leftarrow \mathsf{ukgen}()$. Check that $\mathsf{CheckGskHonest}(gsk) = 1$ and store $\langle \mathcal{M}_i, \mathsf{bsn}, gsk\rangle$ in `DomainKeys`.
    - Compute a signature as $\sigma \leftarrow \mathsf{sig}(gsk, m, \mathsf{bsn}, \hat{p})$ and check $\mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p}) = 1$.
    - Check that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk) = 1$ and check that there is no $\mathcal{M}'_i \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$.
  - If $\mathcal{M}_i$ is honest, store $\langle m, \mathsf{bsn}, \sigma, \mathcal{M}_i, \hat{p}\rangle$ in `Signed`.
  - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

13. `Verify`. On input $(\mathsf{VERIFY}, sid, m, \mathsf{bsn}, \sigma, \hat{p}, \mathtt{RL})$ from some party $\mathcal{V}$.
  - Retrieve all pairs $(\mathcal{M}_i, gsk_i)$ from $\langle \mathcal{M}_i, *, gsk_i, *\rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i\rangle \in$ `DomainKeys` such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
    - More than one key gsk_i was found.
    - $\mathcal{I}$ is honest and no pair $(\mathcal{M}_i, gsk_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs\rangle \in$ `Members` with $\hat{p}(attrs) = 1$ exists.
    - There is an honest $\mathcal{M}_i$ but no entry $\langle m, \mathsf{bsn}, *, \mathcal{M}_i, \hat{p}\rangle \in$ `Signed` exists.
    - There is a $gsk' \in \mathtt{RL}$ such that $\mathsf{identify}(m, \mathsf{bsn}, \sigma, gsk') = 1$ and no pair $(\mathcal{M}_i, gsk_i)$ for an honest $\mathcal{M}_i$ was found.
  - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(m, \mathsf{bsn}, \sigma, \hat{p})$.
  - Add $\langle m, \mathsf{bsn}, \sigma, \mathtt{RL}, f\rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

14. `Link`. On input $(\mathsf{LINK}, sid, m_0, \sigma_0, \hat{p}_0, m_1, \sigma_1, \hat{p}_1, \mathsf{bsn})$ from some party $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$.
  - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(m_0, \mathsf{bsn}, \sigma_0, \hat{p}_0)$ or $(m_1, \mathsf{bsn}, \sigma_1, \hat{p}_1)$ is not valid, which is verified via the VERIFY interface with $\mathtt{RL} = \emptyset$.
  - For each key gsk_i in `Members` and `DomainKeys`, compute $b_i \leftarrow \mathsf{identify}(m_0, \mathsf{bsn}, \sigma_0, gsk_i)$ and $b'_i \leftarrow \mathsf{identify}(m_1, \mathsf{bsn}, \sigma_1, gsk_i)$, and then do the following:
    - Set $f \leftarrow 0$ if $b_i \neq b'_i$ for some i.
    - Set $f \leftarrow 1$ if $b_i = b'_i = 1$ for some i.
  - If f is not defined yet, set $f \leftarrow \mathsf{link}(m_0, \sigma_0, m_1, \sigma_1, \mathsf{bsn})$.
  - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 29. Functionality $\mathcal{F}$ for Game 17