Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited *

Jan Camenisch[1], Manu Drijvers[1,2], and Anja Lehmann[1]

[1] IBM Research – Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
{jca,mdr,anj}@zurich.ibm.com
[2] Department of Computer Science, ETH Zurich, 8092 Zürich, Switzerland

Abstract. Direct Anonymous Attestation (DAA) is a cryptographic protocol for privacy-protecting authentication. It is standardized in the TPM standard and implemented in millions of chips. A variant of DAA is also used in Intel's SGX. Recently, Camenisch et al. (PKC 2016) demonstrated that existing security models for DAA do not correctly capture all security requirements, and showed a number of flaws in existing schemes based on the LRSW assumption. In this work, we identify flaws in security proofs of a number of qSDH-based DAA schemes and point out that none of the proposed schemes can be proven secure in the recent model by Camenisch et al. (PKC 2016). We therefore present a new, provably secure DAA scheme that is based on the qSDH assumption. The new scheme is one of the most efficient DAA schemes, with support for DAA extensions to signature-based revocation and attributes. We rigorously prove the scheme secure in the model of Camenisch et al., which we modify to support the extensions. As a side-result of independent interest, we prove that the BBS+ signature scheme is secure in the type-3 pairing setting, allowing for our scheme to be used with the most efficient pairing-friendly curves.

1 Introduction

Direct anonymous attestation (DAA) is a cryptographic authentication protocol that lets a platform, consisting of a secure element and a host, create anonymous attestations. These attestations are signatures on messages and convince a verifier that the message was signed by a authorized secure element, while preserving the privacy of the platform. DAA was designed for the Trusted Platform Module (TPM) by Brickell, Camenisch, and Chen [BCC04] and was standardized in the TPM 1.2 specification in 2004 [Tru04]. Their paper inspired a large body of work on DAA schemes [BCL08, CMS08, CF08, BCL09, Che09, CPS10, BL10, BFG+13, CDL16b], including more efficient scheme using bilinear pairings as well as different security definitions and proofs. One result of these works is the

* An extended abstract of this work was published at TRUST 2016 [CDL16a]. This is the full version. This work has been supported by the ERC under Grant PERCY #321310.

recent TPM 2.0 specification [Tru14, Int15] that includes support for multiple pairing-based DAA schemes, two of which are standardized by ISO [Int13].

DAA is widely used in the area of trusted computing. Over 500 million TPMs have been sold[3], making DAA probably the most complex cryptographic scheme that is widely implemented. Additionally, an extension of DAA is used in the Intel Software Guard Extensions (SGX) [CD16], the most recent development in the area of trusted computing.

A number of functional extensions to DAA have been proposed. Brickell and Li [BL07, BL11] introduced Enhanced Privacy ID (EPID), which extends DAA with signature-based revocation. This extension allows one to revoke a platform based on a previous signature from that platform. This is an improvement over the private key revocation used in DAA schemes, where a TPM cannot be revoked without knowing its secret key.

Chen and Urian [CU15] introduced DAA with attributes (DAA-A), in which the membership credential can also contain attributes. These attributes might include more information about the platform, such as the vendor or model, or other information, such as an expiration date of the credential. When signing, the platform can selectively disclose attributes, e.g., reveal that the signature was created by a TPM of a certain manufacturer, or create more advanced proofs, such as proving that the expiration date of the credential lies in the future.

Unfortunately, in spite of being used in practice, many of the existing schemes are not provably secure. Recently, Camenisch et al. [CDL16b] showed that previous security definitions of DAA are not satisfactory, meaning that security proofs using these security models do not guarantee security. They further point out that many of the DAA schemes based on the LRSW assumption [LRSW99] are flawed. They finally provide a comprehensive security model and provide a LRSW-based scheme that is provably secure in their model. However, there is to date no scheme based on the qSDH assumption [BB08] that is secure in their model.

Indeed, in this work we show that also many of the DAA schemes based on the qSDH assumption are flawed. The most efficient qSDH-based schemes [Che09, BL10, CU15] use a credential which is not provably secure against adaptive chosen message attacks, leaving room for an attacker to forge credentials. Moreover, these schemes use a flawed proof-of-knowledge of credentials, which in fact does not prove possession of such a credential. Finally, the security of all existing qSDH-based schemes has only been analyzed in the type-2 pairing setting [GPS08]. However, these schemes are often used in the more efficient type-3 setting, where there is no efficient isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. As the security proofs rely on such an isomorphism, they do not apply to a type-3 setting, meaning there is no evidence of security.

Apart from pointing out flaws in the existing qSDH-based DAA schemes, this paper provides two more main contributions. First, we fix the issues and present a qSDH-based DAA scheme with support for attributes and signature-based revocation. Like previous work, we use the BBS+ signature [ASM06] for

[3] `http://www.trustedcomputinggroup.org/solutions/authentication`

credentials, but unlike previous work we move to the more efficient and flexible type-3 pairing setting. Using a new way to prove knowledge of credentials, our scheme is more efficient than any existing DAA scheme. Second, we extend the security model by Camenisch et al. [CDL16b] to capture signature-based revocation and support attributes, and rigorously prove our scheme secure in this model.

2 Flaws in Existing qSDH-based Schemes

The first DAA scheme by Brickell et al. [BCC04] is based on the strong RSA assumption. Due to the large keys required for RSA, this protocol was inefficient and hard to implement. A lot of research has gone into designing more efficient DAA schemes using bilinear pairings and improving the security model of DAA. The work on efficient DAA schemes can be split in two chains of work, one based on the LRSW assumption [LRSW99], and one on the qSDH assumption [BB04]. The schemes based on the LRSW assumption have recently been studied by Camenisch et al. [CDL16b]. In this section we now discuss the existing qSDH-based schemes and their proofs of security. We start by giving an overview of existing security models for DAA and DAA with extensions, and then show that none of the existing qSDH-based are efficient and provably secure.

2.1 Security Models for DAA

One of the most challenging tasks in cryptography is to formally define a security model that allows for rigorous security proofs. Before we discuss security models, we give some intuition on the required security properties of DAA. First, signatures must be *unforgeable*, meaning only platforms that the issuer allowed to join can create signatures. Second, signatures must be *anonymous*. A basename is used to control anonymity, and an adversary given two signatures valid with respect to two distinct basenames must not be able to decide whether the signatures were created by the same platform. Third, we require *non-frameability*. When a platform signs with respect to the same basename multiple times, a verifier can link these signatures, meaning it realizes both signatures stem from the same platform. No adversary should be able to frame a platform, meaning it cannot create a signature on a message m that links to some platform's signatures, while that platform never signed m.

There are multiple ways to define a security model. Property-based definitions are a set of security games, where every game defines a security property, and a scheme is secure when every property holds. Simulation-based definitions consist of a trusted third party. In a so-called ideal world, every protocol participant hands their inputs to the trusted third party rather than executing the protocol, and outputs are generated by the trusted third party. As the trusted third party performs the task in a way secure by design, the ideal world performs the desired task securely. A protocol is considered secure if the real world, in which protocol participants execute the protocol, is as secure as the ideal world.

The first security model for DAA as introduced by Brickell et al. [BCC04] follows the simulation-based paradigm. Therein, signature generation and verification is modeled as an interactive process, meaning a signature must always be verified immediately and cannot be used further. Camenisch et al. [CDL16b] define a simulation-based security model for DAA that outputs signatures and allows them to be used in any way.

In an attempt to simplify the security model of DAA, Brickell et al. [BCL09] introduce a property-based definition for DAA. Unfortunately, this definition does not cover non-frameability, and the notion for unforgeability allows forgeable schemes to be proven secure: A scheme in which one value is a signature on every message can fulfill the security model, while clearly being insecure. Chen [Che09] extends this definition with a property for non-frameability, but the other issues remain. Brickell and Li create a property-based security model for enhanced privacy ID (EPID) [BL11] very similar to the model of Brickell et al. [BCL09], and containing the same flaws.

Camenisch et al. [CDL16b] give a more detailed overview of the security models for DAA.

2.2 qSDH-based DAA Schemes and Proofs

Chen and Feng [CF08] introduce the first DAA scheme based on the qSDH assumption. The scheme requires the TPM to work in the target group $\mathbb{G}_T$, which is inefficient and makes implementation more involved. Chen [Che09] improves the efficiency of the previous schemes by removing one element of the membership credential. Brickell and Li [BL10] further improve the efficiency by changing the distribution of work between the host and TPM such that the TPM only performs computations in $\mathbb{G}_1$. Being the most efficient scheme, it is supported by the TPM 2.0 standard and ISO standardized [Int13].

All three schemes come with proofs of security using the security models by Brickell et al. [BCL09] and Brickell and Li [BL11]. However, as these models allow one to prove insecure schemes secure, proofs in these models are not actual evidence of security. Furthermore, the proofs of the two most efficient schemes [Che09, BL10] are invalid, as the membership credential is not proven to be existentially unforgeable against adaptive chosen message attacks. The proof aims to reduce a credential forgery to breaking the qSDH assumption, meaning that the issuer private key is an unknown value defined by the qSDH instance. They start by using the Boneh-Boyen trick [BB04] to create $q-1$ weak BB signatures under the issuer key, on previously chosen e_i values. From every weak BB signature, one membership credential on a (potentially adversarial) platform key can be created. For one randomly selected honest platform joining, it returns a credential on a key chosen during the parameter selection of the scheme. It can create this credential without consuming a BB04 signature due to the special selection of parameters. Since the key is chosen like an honest platform would, this simulation is valid for honest platforms. Finally, the authors claim that when a credential forgery occurs that reuses part of an issued credential, with probability $\frac{1}{q}$, it is reusing part of the specially crafted credential. This

is not true, as there may not even be honest platforms joining, or the adversary may disregard credentials issued to honest platforms. To fix the proof, one must be able to issue the special credential also to corrupt platforms, i.e., on a key chosen by the adversary, but this does not seem possible.

Related to this issue, the proofs of knowledge proving knowledge of a credential in these schemes do not prove the correct statement. The prover proves knowledge of TPM secret gsk and of values a, b. The proof only proves knowledge of a valid credential when $b = a \cdot gsk$, but this structure of b is not proven. This means that from a signature that passes verification, one cannot always extract a valid signature, which prevents proving unforgeability. This could be fixed by also proving $b = a \cdot gsk$ in zero knowledge.

Finally, the security proofs of all the pairing-based schemes mentioned here make use of an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ in the security proof. This prevents the schemes from being used with the more efficient type-3 curves [GPS08]. However, the TPM 2.0 standard [Tru14, Int15], designed to support the DAA scheme by Brickell and Li [BL10], uses such type-3 curves. As there is no efficient isomorphism in this setting, any security proof requiring an isomorphism is not applicable, leaving the security of the scheme unproven.

DAA with Extensions. Two extensions of DAA have been proposed. Brickell and Li [BL11] present EPID based on the qSDH assumption. This extends DAA with signature-based revocation, allowing revocation of platforms based on a signature from that platform. Unfortunately, they do not show how the work of the platform can be split between a TPM and host. Chen and Urian [CU15] introduce DAA with attributes (DAA-A), where the membership credential does not only contain the TPM key, but also attribute values. This allows for many new use cases, such as showing that a signature was created by a platform of a certain vendor, or adding expiration dates to credentials. The authors present two instantiations, one based on the LRSW assumption and one based on the qSDH assumption. Unfortunately, the schemes do not come with security proofs. The qSDH scheme suffers from the same flaws as the most recent qSDH DAA schemes discussed above, i.e., the credential is not proven to be unforgeable. Worse, the LRSW scheme is forgeable using the trivial credential $A = B = C = D = E_1 = \ldots = E_L = 1_{\mathbb{G}_1}$ that signs all attributes and keys, so anyone can sign with respect to any desired set of attributes.

3 A New Security Model For DAA With Extensions

In this section we present our security model for DAA with attributes and signature-based revocation, which is defined as an ideal functionality $\mathcal{F}^l_{\mathsf{daa}+}$ in the UC framework [Can01]. In UC, an environment $\mathcal{E}$ passes inputs and outputs to the protocol parties. The network is controlled by an adversary $\mathcal{A}$ that may communicate freely with $\mathcal{E}$. In the ideal world, the parties forward their inputs to the ideal functionality $\mathcal{F}$, which then (internally) performs the defined task and creates outputs that the parties forward to $\mathcal{E}$. Roughly, a real-world protocol

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 (ALG, sid, sig, ver, link, identify, ukgen) from $\mathcal{S}$
   - Check that ver, link and identify are deterministic **(i)**.
   - Store (sid, sig, ver, link, identify, ukgen) and output (SETUPDONE, sid) to $\mathcal{I}$.

Join

3. `Join Request.` On input (JOIN, $sid, jsid, \mathcal{M}_i$) from host $\mathcal{H}_j$.
   - Create a join session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, \perp, status\rangle$ 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 **(ii)**.
   - Output (JOINPROCEED, $sid, jsid, \mathcal{M}_i$) to $\mathcal{I}$.
5. `Join Proceed.` On input (JOINPROCEED, $sid, jsid, attrs$) from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ to $status \leftarrow complete$.
   - 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 by checking
     - CheckGskHonest(gsk) $= 1$ **(iii)** if $\mathcal{H}_j$ is corrupt and $\mathcal{M}_i$ is honest, or
     - CheckGskCorrupt(gsk) $= 1$ **(iv)** if $\mathcal{M}_i$ is corrupt.
   - Insert $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Members` and output (JOINED, $sid, jsid$) to $\mathcal{H}_j$.

Fig. 1. The Setup and Join related interfaces of $\mathcal{F}^l_{\mathsf{daa}+}$. *(The roman numbers are labels for the different checks made within the functionality and will be used as references in the analysis of the functionality and the proof.)*

Π is said to securely realize a functionality $\mathcal{F}$, if the real world is indistinguishable from the ideal world, meaning for every adversary performing an attack in the real world, there is an ideal world adversary (often called simulator) $\mathcal{S}$ that performs the same attack in the ideal world.

3.1 Ideal Functionality $\mathcal{F}^l_{\mathsf{daa}+}$

We now formally define our ideal functionality $\mathcal{F}^l_{\mathsf{daa}+}$, which is a modification of $\mathcal{F}^l_{\mathsf{daa}}$ as defined by Camenisch et al. [CDL16b]. The modifications extend the functionality to support signature-based revocation and attributes.

The UC framework allows us to focus our analysis on a single protocol instance with a globally unique session identifier sid. Here we use session identifiers of the form $sid = (\mathcal{I}, sid')$ for some issuer $\mathcal{I}$ and a unique string sid'. To allow several sub-sessions for the join and sign related interfaces we use unique subsession identifiers $jsid$ and $ssid$. Our ideal functionality $\mathcal{F}^l_{\mathsf{daa}+}$ is parametrized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$, that we need to model the information leakage that occurs in the communication between a host $\mathcal{H}_i$ and TPM $\mathcal{M}_j$. As our functionality supports attributes, we have parameters L and $\{\mathbb{A}_i\}_{0<i\leq L}$,

Sign

7. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
8. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
9. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ 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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that there are no two distinct gsk values matching σ' **(v)**.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found **(vi)**.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk \rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$ **(vii)**, and store $\langle \mathcal{M}_i, bsn, gsk \rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$ **(viii)**.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ **(ix)** and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$ **(x)**.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

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

Link

12. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$) **(xvii)**.
   - For each gsk_i in `Members` and `DomainKeys` compute $b_i \leftarrow \mathsf{identify}(\sigma, m, bsn, gsk_i)$ and $b_i' \leftarrow \mathsf{identify}(\sigma', m', bsn, gsk_i)$ and do the following:
     - Set $f \leftarrow 0$ if $b_i \neq b_i'$ for some i **(xviii)**.
     - Set $f \leftarrow 1$ if $b_i = b_i' = 1$ for some i **(xix)**.
   - If f is not defined yet, set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 2. The Sign, Verify, and Link related interfaces of $\mathcal{F}_{\mathsf{daa}+}^{l}$

where L is the amount of attributes every credential contains and $\mathbb{A}_i$ the set from which the i-th attribute is taken. A parameter $\mathbb{P}$ is used to describe which proofs over the attributes platforms can make. This generic approach lets the functionality capture both simple protocols that only support selective disclosure and more advanced protocols that support arbitrary predicates. Every element $p \in \mathbb{P}$ is a predicate over the attributes: $\mathbb{A}_1 \times \ldots \times \mathbb{A}_L \to \{0, 1\}$.

The full definition of $\mathcal{F}^{l}_{\mathsf{daa+}}$ is presented in Fig. 1 and Fig. 2. The changes with respect to the DAA functionality by Camenisch et al. [CDL16b] are highlighted in blue. Two macros are used to simplify the presentation of the functionality:

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

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

Camenisch et al. [CDL16b] give an extensive argumentation of why their functionality guarantees the desired properties. We now argue that our changes indeed allow for attributes and signature-based revocation and that they do not have a negative impact on the other properties guaranteed by the functionality.

Attributes. The issuer is in charge of the attributes, and must explicitly allow a platform to be issued certain attributes with the JOINPROCEED output and input. The verification interface now checks whether the signer has the correct attributes, fulfilling the attribute predicate (Check **(xii)**). This guarantees that no platform can create valid signatures with respect to attribute predicates that do not hold for the attributes of this platform.

Signature-based Revocation. The sign interface now takes a signature-based revocation list `SRL` as input. The functionality does not sign for platforms that are revoked by `SRL`, which it enforces via Check **(vi)**. Further, the verification interface will reject signatures from platforms revoked in `SRL` by checking whether any of those signatures is based on the key gsk from the signature being verified.

Our functionality enforces that every signature matches to only one gsk value. To ensure this also for the signatures specified in `SRL`, Check **(v)** has been added and the `CheckGsk` macros have been extended to also take the `SRL` values into consideration.

4 Building Blocks

In this section we introduce the building blocks used by our construction. In addition to the standard building blocks such as bilinear pairings and the qSDH assumption, we introduce the BBS+ signature without requiring an isomorphism between the bilinear groups. Up to now, this signature has only been proven secure using such an isomorphism, limiting the settings in which the signature can be used. Furthermore, we show how to efficiently prove knowledge of a BBS+ signature.

4.1 Bilinear Maps

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be groups of prime order p. A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ must satisfy bilinearity, i.e., $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$; non-degeneracy, i.e., for all generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, $e(g_1, g_2)$ generates $\mathbb{G}_T$; and efficiency, i.e., there exists an efficient algorithm $\mathcal{G}(1^\tau)$ that outputs the bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ and an efficient algorithm to compute $e(a, b)$ for any $a \in \mathbb{G}_1, b \in \mathbb{G}_2$.

Galbraith et al. [GPS08] distinguish three types of pairings: type-1, in which $\mathbb{G}_1 = \mathbb{G}_2$; type-2, in which $\mathbb{G}_1 \neq \mathbb{G}_2$ and there exists an efficient isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$; and type-3, in which $\mathbb{G}_1 \neq \mathbb{G}_2$ and no such isomorphism exists.

Type-3 pairings currently allow for the most efficient operations in $\mathbb{G}_1$ given a security level using BN curves with a high embedding degree [BN06]. Therefore it is desirable to describe a cryptographic scheme in a type-3 setting, i.e., without assuming $\mathbb{G}_1 = \mathbb{G}_2$ or the existence of an efficient isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$.

4.2 *q*-Strong Diffie-Hellman Assumption

The q-Strong Diffie-Hellman (qSDH) problem has two versions. The first version by Boneh and Boyen is defined in a type-1 and type-2 pairing setting [BB04]. This version, to which we refer as the Eurocrypt version, is informally stated as follows:

> Given a q+2-tuple $(g_1, g_2, g_2^x, g_2^{(x^2)}, \ldots, g_2^{(x^q)}) \in \mathbb{G}_1 \times \mathbb{G}_2^{q+1}$ with $g_1 = \psi(g_2)$, output a pair $(c, g_1^{1/(x+c)}) \in \mathbb{Z}_p^* \times \mathbb{G}_1$.

Boneh and Boyen created a new version of the qSDH problem to support type-3 settings [BB08]. The so-called JOC version is informally stated as follows:

> Given a q+3-tuple $(g_1, g_1^x, g_1^{(x^2)}, \ldots, g_1^{(x^q)}, g_2, g_2^x) \in \mathbb{G}_1^{q+1} \times \mathbb{G}_2^2$, output a pair $(c, g_1^{1/(x+c)}) \in \mathbb{Z}_p \setminus \{-x\} \times \mathbb{G}_1$.

4.3 BBS+ Signatures

We recall the BBS+ signature, as described by Au et al. [ASM06], which is inspired by the group signature scheme by Boneh et al. [BBS04].

Key Generation Take $(h_0, \ldots, h_L) \xleftarrow{\$} \mathbb{G}_1^{L+1}$, $x \xleftarrow{\$} \mathbb{Z}_p^*$, $w \leftarrow g_2^x$, and set $sk = x$ and $pk = (w, h_0, \ldots, h_L)$.

Signature On input message $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ and secret key x, pick $e, s \xleftarrow{\$} \mathbb{Z}_p$ and compute $A \leftarrow (g_1 h_0^s \prod_{i=1}^{L} h_i^{m_i})^{\frac{1}{e+x}}$. Output signature $\sigma \leftarrow (A, e, s)$.

Verification On input a public key $(w, h_0, \ldots, h_L) \in \mathbb{G}_2 \times \mathbb{G}_1^{L+1}$, message $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$, and purported signature $(A, e, s) \in \mathbb{G}_1 \times \mathbb{Z}_p^2$, check $e(A, w g_2^e) = e(g_1 h_0^s \prod_{i=1}^{L} h_i^{m_i}, g_2)$.

Au et al. prove the BBS+ signature secure under the Eurocrypt version of the qSDH assumption, making use of the isomorphism between the groups in the security proof. As in type-3 pairings no such isomorphism exists, this means the proof is not valid when this isomorphism does not exist and we do not know whether the signature is secure in this setting. We modify the proof by Au et al. to use the JOC version of the qSDH assumption and no longer rely on an isomorphism in the proof, allowing us to use BBS+ signatures with type-3 pairings.

Lemma 1. *The BBS+ signature scheme is existentially unforgeable against adaptive chosen message attacks under the JOC version of the qSDH assumption, in particular in pairing groups where no efficient isomorphism between $\mathbb{G}_2$ and $\mathbb{G}_1$ exists.*

Proof. We reduce the security of the BBS+ signature to the (JOC version) qSDH assumption, by defining an adversary $\mathcal{A}'$ that breaks the qSDH assumption given a BBS+ forger $\mathcal{A}$. The reduction does not assume $\mathbb{G}_1 = \mathbb{G}_2$ or the existence of an efficient isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, meaning that the proof also holds in type 3 pairing settings. This proof is based on the proof by Au et al. [ASM06], which does rely on such an isomorphism.

$\mathcal{A}$' receives $(d_0 = g_1, d_1 = g_1^x, d_2 = g_1^{x^2}, \ldots, d_q = g_1^{x^q}, g_2, w = g_2^x) \in \mathbb{G}_1^{q+1} \times \mathbb{G}_2^2$ and chooses $q-1$ values $e_i \xleftarrow{\$} \mathbb{Z}_p$. We use the Boney-Boyen technique [BB08] to create weak BB signatures by defining polynomial

$$f(X) = \prod_{i=1}^{q-1}(X + e_i) = \sum_{i=0}^{q-1} \alpha_i X^i (\text{for some easily computable } \alpha_i) \tag{1}$$

$\mathcal{A}$' defines g_1', d_1' as follows.

$$g_1' = \prod_{i=0}^{q-1} d_i^{\alpha_i \theta} = g_1^{\theta f(x)} \tag{2}$$

$$d_1' = \prod_{i=0}^{q-1} {d_{i+1}}^{\alpha_i \theta} = \prod_{i=0}^{q-1} ({d_i}^x)^{\alpha_i \theta} = (g_1')^x \tag{3}$$

We will need $d_1' = {g_1'}^x$ later in the proof. Note that in the computation of d_1', we use $g_1^{x^q}$, while for the computation of g_1', we only need the d-values up to $g_1^{x^{q-1}}$. This means we can only simulate $q-1$ rather than q signatures under

the q-SDH assumption. Previous works considered a type 2 pairing which has a isomorphism ψ that lets you compute this element without requiring an higher q.

$\mathcal{A}$' now creates weak BB signatures on every e_i by taking polynomial

$$\begin{aligned} f_i(X) &= f(X)/(X+e_i) \\ &= \prod_{j=1, j\neq i}^{q-1} (X+e_i) \\ &= \sum_{j=0}^{q-1} \beta_j X^j (\text{for some easily computable } \beta_j) \end{aligned} \tag{4}$$

and computing weak BB signature

$$B_i = \prod_{j=0}^{q-2} d_j^{\beta_j \theta} = g_1^{\theta f_i(x)} = {g'_1}^{1/(x+e_i)} \tag{5}$$

Now $\mathcal{A}$' must answer the adaptive signing queries from $\mathcal{A}$. $\mathcal{A}$' chooses $e, a, k \xleftarrow{\$} \mathbb{Z}_p^*$ and computes

$$\begin{aligned} h_0 &= ((d'_1 {g'_1}^e)^k {g'_1}^{-1})^{1/a} \\ &= (({g'_1}^x {g'_1}^e)^k {g'_1}^{-1})^{1/a} \quad \text{(by (3))} \\ &= {g'_1}^{\frac{(e+x)k-1}{a}} \end{aligned} \tag{6}$$

The other generators $h_1, \ldots, h_l$ are set by taking $(\mu_1, \ldots, \mu_l) \xleftarrow{\$} \mathbb{Z}_q^l$ and setting

$$h_i \leftarrow h_0^{\mu_i} \tag{7}$$

$\mathcal{A}$' outputs public key $g'_1, h_0, \ldots, h_l, g_2, w$ and lets $\mathcal{A}$ make q signing queries. $\mathcal{A}$' picks $\alpha \xleftarrow{\$} \mathbb{Z}_q$, and the α-th query will be treated in a special way. The special query on message $(m_{\alpha,1}, \ldots, m_{\alpha,l})$ is treated as follows. $\mathcal{A}$' sets

$$s_\alpha \leftarrow a - \sum_{j=1}^{l} \mu_j \cdot m_{\alpha,j} \tag{8}$$

and outputs signature $(A_\alpha \leftarrow {g_1'}^k, e, s_\alpha)$. This is a valid signature:

$$\begin{aligned}
A_\alpha &= {g_1'}^k \\
&= ({g_1'}^{(e+x)k})^{\frac{1}{e+x}} \\
&= (g_1' \cdot {g_1'}^{(e+x)k-1})^{\frac{1}{e+x}} \\
&= (g_1' \cdot h_0^a)^{\frac{1}{e+x}} \quad \text{(by (6))} \\
&= (g_1' \cdot h_0^{s_\alpha + \sum_{j=1}^{l} \mu_j \cdot m_{\alpha,j}})^{\frac{1}{e+x}} \quad \text{(by (8))} \\
&= (g_1' \cdot h_0^{s_\alpha} \cdot \prod_{j=1}^{l} h_j^{m_{\alpha,j}})^{\frac{1}{e+x}}
\end{aligned} \tag{9}$$

All other signing queries are handled by consuming one of the BB signatures. On a message $(m_{i,1}, \ldots, m_{i,l})$, $\mathcal{C}$ takes a BB signature (B_i, e_i) and chooses a random $s_i \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{A}$' computes $A_i \leftarrow B_i \cdot (B_i^{\frac{(e-e_i)\cdot k-1}{a}} \cdot {g_1'}^{\frac{k}{a}})^{s_i + \sum_{j=0}^{l} \mu_j \cdot m_{i,j}}$ and outputs signature (A_i, e_i, s_i). This is a valid signature:

$$\begin{aligned}
A_i &= B_i \cdot \left(B_i^{\frac{(e-e_i)\cdot k-1}{a}} \cdot {g_1'}^{\frac{k}{a}}\right)^{s_i + \sum_{j=0}^{l} \mu_j \cdot m_{i,j}} \\
&= B_i \cdot \left(B_i^{\frac{(e-e_i)\cdot k-1}{a}} \cdot B_i^{\frac{(x+e_i)k}{a}}\right)^{s_i + \sum_{j=0}^{l} \mu_j \cdot m_{i,j}} \quad \text{(by (5))} \\
&= B_i \cdot B_i^{\frac{(s_i + \sum_{j=0}^{l} \mu_j \cdot m_{i,j}) \cdot (e+x) \cdot k-1}{a}} \\
&= B_i \cdot \left(g_1'^{\frac{(s_i + \sum_{j=0}^{l} \mu_j \cdot m_{i,j}) \cdot (e+x) \cdot k-1}{a}}\right)^{\frac{1}{x+e_i}} \\
&= B_i \cdot \left(h_0^{s_i + \sum_{j=0}^{l} \mu_j \cdot m_{i,j}}\right)^{\frac{1}{x+e_i}} \\
&= B_i \cdot \left(h_0^{s_i} \cdot \prod_{j=1}^{l} h_j^{m_{i,j}}\right)^{\frac{1}{x+e_i}} \\
&= g_1'^{\frac{1}{x+e_i}} \cdot \left(h_0^{s_i} \cdot \prod_{j=1}^{l} h_j^{m_{i,j}}\right)^{\frac{1}{x+e_i}} \quad \text{(by (5))} \\
&= \left(g_1' \cdot h_0^{s_i} \cdot \prod_{j=1}^{l} h_j^{m_{i,j}}\right)^{\frac{1}{x+e_i}}
\end{aligned} \tag{10}$$

Finally, with non-negligible probability $\mathcal{A}$ submits a valid forgery (A^*, e^*, s^*) on message $(m_1^*, \ldots, m_l^*)$. We distinguish three cases.

– Case $e^* \notin \{e_i\}_{0<i\leq q, i\neq\alpha} \cup \{e\}$: By setting $y \leftarrow s^* + m_1^*\mu_1 + \ldots + m_L^*\mu_L$ and $B^* \leftarrow (A^* {g_1'}^{\frac{-ky}{a}})^{\frac{a}{a-y-ky(e^*-e)}}$, $\mathcal{A}$' gets an extra SDH pair.

$$\begin{aligned}
B^* &= (A^* {g_1'}^{\frac{-ky}{a}})^{\frac{a}{a-y-ky(e^*-e)}} \\
&= {A^*}^{\frac{a}{a-y-ky(e^*-e)}} {g_1'}^{\frac{-ky}{a-y-ky(e^*-e)}} \\
&= (g_1' h_0^y)^{\frac{a}{(e^*+x)(a-y-ky(e^*-e))}} {g_1'}^{\frac{-ky}{a-y-ky(e^*-e)}} \\
&= (g_1' {g_1'}^{\frac{y(e^*+x)k-y}{a}})^{\frac{a}{(x+e^*)(a-y-ky(e^*-e))}} {g_1'}^{\frac{-ky}{a-y-ky(e^*-e)}} \\
&= ({g_1'}^{\frac{y(e^*+x)k-y+a}{a}})^{\frac{a}{(x+e^*)(a-y-ky(e^*-e))}} {g_1'}^{\frac{-ky}{a-y-ky(e^*-e)}} \\
&= {g_1'}^{\frac{y(e^*+x)k-y+a}{(x+e^*)(a-y-ky(e^*-e))}} {g_1'}^{\frac{-ky}{a-y-ky(e^*-e)}} \\
&= {g_1'}^{\frac{y(e^*+x)k-y+a+-ky(x+e^*)}{(x+e^*)(a-y-ky(e^*-e))}} \\
&= {g_1'}^{\frac{a-y-ky(e^*-e)}{(x+e^*)(a-y-ky(e^*-e))}} \\
&= {g_1'}^{\frac{1}{x+e^*}}
\end{aligned} \tag{11}$$

From an extra SDH pair $\mathcal{A}$' can break the qSDH assumption as shown in [BB08].

– Case $e^* = e_i$ and $A^* = A_i$ but $(m_1^*, \ldots, m_L^*) \neq (m_{i,1}, \ldots, m_{i,L})$, then $\mathcal{A}$' can break the DL assumption (which is implied by the qSDH assumption). For some i', $\mathcal{A}$ finds $m_{i'}^* \neq m_{i',j}$ with nonnegligible probability. We set a discrete logarithm instance as our $h_{i'}$ generator and, from this forgery, learn $log_{h_0}(h_{i'}) = \frac{s-s^*+\sum_{j=1,\ldots,L, j\neq i'} \mu_i(m_{i,j}-m_j^*)}{m_{i'}^*-m_{i,i'}}$.

– Case $e^* \in \{e_i\}_{0<i\leq q, i\neq\alpha} \cup \{e\}$: With probability $\frac{1}{q}$, $e^* = e$. We can derive an extra SDH pair (B^*, e^*) in the exact same way as in the first case. □

4.4 Proof Protocols

When referring to the zero-knowledge proofs of knowledge of discrete logarithms and statements about them, we will follow the notation introduced by Camenisch and Stadler [CS97] and formally defined by Camenisch, Kiayias, and Yung [CKY09].

For instance, $PK\{(a,b,c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$ denotes a "*zero-knowledge proof of knowledge of integers a, b and c such that* $y = g^a h^b$ *and* $\tilde{y} = \tilde{g}^a \tilde{h}^c$ *holds*," where $y, g, h, \tilde{y}, \tilde{g}$ and $\tilde{h}$ are elements of some groups $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\tilde{\mathbb{G}} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$. Given a protocol in this notation, it is straightforward to derive an actual protocol implementing the proof [CKY09]. Indeed, the computational complexities of the proof protocol can be easily derived from this notation: for each term $y = g^a h^b$, the prover and the verifier have to perform an equivalent computation, and to transmit one group element and one response value for each exponent.

SPK denotes a signature proof of knowledge, that is a non-interactive transformation of a proof with the Fiat-Shamir heuristic [FS87] in the random oracle

model [BR93]. From these non-interactive proofs, the witness can be extracted by rewinding the prover and programming the random oracle. Alternatively, these proofs can be extended to be online-extractable, by verifiably encrypting the witness to a public key defined in the common reference string (CRS). A practical instantiation is given by Camenisch and Shoup [CS03] using Paillier encryption, secure under the DCR assumption [Pai99].

4.5 Proving Knowledge of a BBS+ Signature

Au et al. [ASM06] show how to prove knowledge of a BBS+ signature, and similar constructions are used in qSDH-based DAA schemes [Che09, BL10]. These proofs take place in $\mathbb{G}_T$, which means computation is slow, group elements are large, and implementations are more complex. We present a new proof of knowledge for BBS+ signatures that takes place only in $\mathbb{G}_1$, which is similar to the efficient proof of knowledge for weak Boneh-Boyen signatures in recent work by Camenisch et al. [CDH16].

The prover has signature $\sigma \leftarrow (A, e, s)$ with $A = (g_1 h_0^s \prod_{i=1}^{L} h_i^{m_i})^{\frac{1}{e+x}}$. He can prove knowledge of a BBS+ signature while selectively disclosing messages m_i with $i \in D$. Randomize the credential by taking $r_1 \xleftarrow{\$} \mathbb{Z}_p^*$, set $A' \leftarrow A^{r_1}$, and set $r_3 \leftarrow \frac{1}{r_1}$. Set $\bar{A} \leftarrow A'^{-e} \cdot b^{r_1} (= A'^{x})$. Choose $r_2 \xleftarrow{\$} \mathbb{Z}_p$, set $d \leftarrow (g_1 h_0^s \prod_{i=1}^{L} h_i^{m_i})^{r_1} \cdot h_0^{-r_2}$, and set $s' \leftarrow s - r_2 \cdot r_3$. The prover now proves

$$\pi \in SPK\{(\{m_i\}_{i \notin D}, e, r_2, r_3, s') : \bar{A}/d = A'^{-e} \cdot h_0^{r_2} \ \wedge \ g_1 \prod_{i \in D} {h_i}^{m_i} = d^{r_3} h_0^{-s'} \prod_{i \notin D} {h_i}^{-m_i}\}.$$

The resulting proof consists of $(A', \bar{A}, d, \pi)$. To verify a proof, the verifier checks $A' \neq 1_{\mathbb{G}_1}$, $e(A', X) = e(\bar{A}, g_2)$, and verifies π.

Note that our construction does not work for signatures with $A = 1_{\mathbb{G}_1}$, but this happens only with negligible probability for an honest signer. In addition, we require the signer to publish a pair $(\bar{g}_1, \bar{g}_2 = \bar{g}_1^x) \in \mathbb{G}_1^2$ with $\bar{g} \neq 1_{\mathbb{G}_1}$.

Lemma 2. *Our construction forms a zero knowledge proof of knowledge of a BBS+ signature for signatures with $A \neq 1_{\mathbb{G}_1}$.*

Proof. Completeness trivially holds for signatures with $A \neq 1_{\mathbb{G}_1}$.

The proof is zero knowledge: a proof can be simulated using pair $(\bar{g}_1, \bar{g}_2)$ by taking $\rho \xleftarrow{\$} \mathbb{Z}_p^*$ and setting $A' \leftarrow \bar{g}_1^{\rho}$ and $\bar{A} \leftarrow \bar{g}_2^{\rho}$. Take $d \xleftarrow{\$} \mathbb{G}_1$. Note that A', $\bar{A}$, and d are simulated exactly as in a real proof. Simulate π to complete the simulated proof.

The proof is a proof of knowledge, meaning we can extract a BBS+ signature from a zero knowledge proof $(A', \bar{A}, d, \pi)$. Extract $(\{m_i\}_{i \notin D}, e, r_2, r_3, s')$ from π. In case $r_3 = 0$, from the second equation of π we have $g_1 h_0^s \prod_i {h_i}^{m_i} = 1_{\mathbb{G}_1}$, so we can extract a valid signature $(1_{\mathbb{G}_1}, e, s)$. In case $r_3 \neq 0$, by the first equation of π we have $\bar{A} = A'^{-e} \cdot d \cdot h_0^{r_2}$, and as $e(A', X) = e(\bar{A}, g_2)$, we have

$A'^{x+e} = d \cdot h_0^{r_2}$. By the second equation of π and $r_3 \neq 0$, we can rewrite this to $A'^{x+e} = (g_1 h_0^{s+r_2 r_3} \prod_i {h_i}^{m_i})^{\frac{1}{r_3}}$, so $(A'^{r_3}, e, s + r_2 r_3)$ is a valid signature on message $(m_1, \ldots, m_L)$. □

5 Construction

In this section, we present our DAA protocol with attributes and signature-based revocation called $\Pi_{\mathsf{daa}+}$. On a high level, it is similar to previous work on qSDH-based DAA. A platform, consisting of a TPM and a host, must once run the join protocol before it can create signatures. In the join protocol, the TPM authenticates to the issuer. The issuer can decide whether the TPM is allowed to join, and if so, it creates a credential for the platform. The credential is a BBS+ signature on a commitment to the TPM chosen secret key *gsk*, and on attribute values as determined by the issuer. Note that the issuer can choose the attribute values, as we expect the issuer to issue only credentials containing attributes where it knows the 'correct' attribute values, such as the model or vendor of the TPM (which it knows as the TPM authenticated), or an expiration date of the credential. After receiving a credential, the platform can sign a message m by creating a signature proof-of-knowledge proving that it has a credential. A basename *bsn* controls linkability. Choosing a fresh *bsn* yields a signature that cannot be linked to any signature that the platform previously generated, meaning the platform can be fully anonymous. Only when it chooses to reuse a basename, the signatures based on the same basename can be linked, i.e., a verifier can notice that they stem from the same platform. The platform also chooses which attributes it will disclose to a verifier.

Our protocol is parametrized by L, the amount of attributes a credential contains, attribute sets $\mathbb{A}_1, \ldots, \mathbb{A}_L$, and l, the leakage of the secure channels used. For simplicity of the presentation, we describe our construction supporting only selective disclosure as attribute predicates, although it is simple to see how the construction can be extended to allow for more advanced predicates using standard proof techniques. We describe the predicates using a set $D \subseteq \{1, \ldots, L\}$ indicating which attributes are disclosed, and a tuple $I = (a_1, \ldots, a_L)$ setting the desired attribute values. For example, the predicate $D \leftarrow \{2\}$, $I = (\perp, 123, \perp)$ is only true for platforms with credentials in which the second attribute value equals 123. Let $\bar{D} = \{1, \ldots, L\} \setminus D$ be the set of undisclosed attributes.

We assume that a common reference string functionality $\mathcal{F}_{\mathsf{crs}}$ and a certificate authority functionality $\mathcal{F}_{\mathsf{ca}}$ are available to all parties. $\mathcal{F}_{\mathsf{crs}}$ will be used to provide the protocol participants with the system parameters consisting of a security parameter τ, a bilinear group $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p with generators $g_1, h_0, \ldots, h_L$ of $\mathbb{G}_1$ and g_2 of $\mathbb{G}_2$ and bilinear map e, generated via $\mathcal{G}(1^\tau)$. $\mathcal{F}_{\mathsf{ca}}$ allows the issuer to register his public key. We further use random oracles $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}_1$ that is used for the computation of pseudonyms and $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\tau$ which is used for the Fiat-Shamir heuristic in the zero-knowledge proofs.

The TPM and issuer must have an authenticated communication channel in the join protocol. This can be achieved in multiple ways, we abstract away from this by using an ideal functionality for this authenticated channel. The channel is established through the host, who forwards the messages and can append unauthenticated content or block the communication. Thus, the standard $\mathcal{F}_{\mathsf{auth}}$ does not capture the desired security and we use $\mathcal{F}_{\mathsf{auth}*}$ instead that was introduced by Camenisch et al. [CDL16b] specifically for this type of authenticated channel. The communication between a TPM and host is modeled using secure message transmission functionality $\mathcal{F}^{l}_{\mathsf{smt}}$. For definitions of the standard functionalities $\mathcal{F}_{\mathsf{crs}}, \mathcal{F}_{\mathsf{ca}}$ and $\mathcal{F}^{l}_{\mathsf{smt}}$ we refer to [Can01, Can04].

For the sake of readability, we will not explicitly call $\mathcal{F}^{l}_{\mathsf{smt}}$ for communication between a TPM and host, nor write down that parties query $\mathcal{F}_{\mathsf{crs}}$ and $\mathcal{F}_{\mathsf{ca}}$ to retrieve the system parameters and the issuer public key. When a party receives an input or message it does not expect, e.g., protocol messages received out of order, or any of the protocol checks fails, the protocol outputs with failure message $\perp$. For efficiency, a host should precompute values $e(g_1, g_2)$ and $e(h_0, w)$ after joining and a verifier should in addition precompute $e(h_i, g_2)$ for $i = 0, \ldots, L$ to minimize the number of pairing computations, but for readability we write the full pairing function.

5.1 Our DAA Protocol with Extensions $\Pi_{\mathsf{daa}+}$

`Issuer Setup.` In the setup phase, the issuer $\mathcal{I}$ creates a key pair of the BBS+ signature scheme and registers the public key with $\mathcal{F}_{\mathsf{ca}}$.

1. $\mathcal{I}$ upon input (SETUP, sid) generates his key pair:
   - Check that $sid = (\mathcal{I}, sid')$ for some sid'.
   - Choose $x \xleftarrow{\$} \mathbb{Z}_p$ and set $w \leftarrow g_2^x$. Take $\bar{g}_1 \xleftarrow{\$} \mathbb{G}_1^*$ and set $\bar{g}_2 \leftarrow \bar{g}_1^x$. Prove knowledge of the private key by creating $\pi \xleftarrow{\$} SPK\{x : w = g_2^x \wedge \bar{g}_2 = \bar{g}_1^x\}$.
   - Initiate $\mathcal{L}_{\mathsf{JOINED}} \leftarrow \emptyset$.
   - Register the public key (w, π) at $\mathcal{F}_{\mathsf{ca}}$, and store the secret key x.
   - Output $(\mathsf{SETUPDONE}, sid)$.

   Protocol participants will verify π upon retrieving ipk.

`Join Request.` The join protocol runs between the issuer $\mathcal{I}$ and a platform, consisting of a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$. The platform authenticates to the issuer and, if the issuer allows the platform to join with certain attributes, obtains a credential that subsequently enables the platform to create signatures. A unique sub-session identifier $jsid$ distinguishes several join sessions that might run in parallel.

1. $\mathcal{H}_j$ upon input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ parses $sid = (\mathcal{I}, sid')$ and sends the message $(\mathsf{JOIN}, sid, jsid)$ to $\mathcal{I}$.
2. $\mathcal{I}$ upon receiving $(\mathsf{JOIN}, sid, jsid)$ from a party $\mathcal{H}_j$ chooses a fresh nonce $n \xleftarrow{\$} \{0,1\}^{\tau}$ and sends $(sid, jsid, n)$ back to $\mathcal{H}_j$.

3. $\mathcal{H}_j$ upon receiving $(sid, jsid, n)$ from $\mathcal{I}$, sends $(sid, jsid, n)$ to $\mathcal{M}_i$.
4. $\mathcal{M}_i$ upon receiving $(sid, jsid, n)$ from $\mathcal{H}_j$, generates its secret key:
   - Check that no key record exists.
   - Choose $gsk \xleftarrow{\$} \mathbb{Z}_p$ and store the key as $(sid, \mathcal{H}_j, gsk, \perp)$.
   - Set $Q \leftarrow h_1^{gsk}$ and compute $\pi_1 \xleftarrow{\$} SPK\{(gsk) : Q = h_1^{gsk}\}(n)$.
   - Store key record $(sid, \mathcal{H}_j, gsk)$.
   - Send (Q, π_1) via the host to $\mathcal{I}$ using $\mathcal{F}_{\mathsf{auth}*}$.
5. $\mathcal{H}_j$ notices $\mathcal{M}_i$ sending (Q, π_1) over $\mathcal{F}_{\mathsf{auth}*}$ to the issuer, it appends its own identity in the unauthenticated part of the message and forwards the full message to the issuer. It also keeps state as $(jsid, Q)$.
6. $\mathcal{I}$ upon receiving (Q, π_1) authenticated by $\mathcal{M}_i$ and identity $\mathcal{H}_j$ unauthenticated over $\mathcal{F}_{\mathsf{auth}*}$, it verifies π_1 and checks that $\mathcal{M}_i \notin \mathcal{L}_{\mathsf{JOINED}}$. It stores $(jsid, Q, \mathcal{M}_i, \mathcal{H}_j)$ and outputs $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$.

`Join Proceed.` The join session is completed when the issuer receives an explicit input telling him to proceed with join session $jsid$ and issue attributes $attrs = (a_1, \ldots, a_L)$.

1. $\mathcal{I}$ upon input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ generates the BBS+ credential:
   - Retrieve the record $(jsid, Q, \mathcal{M}_i, \mathcal{H}_j)$ and add $\mathcal{M}_i$ to $\mathcal{L}_{\mathsf{JOINED}}$.
   - Choose random $e, s \in \mathbb{Z}_p$.
   - $A \leftarrow (g_1 \cdot h_0^s \cdot Q \cdot \prod_{i=1}^{L} h_{i+1}^{a_i})^{1/(e+x)}$
   - Send the credential to the host by sending $(sid, jsid, A, e, s, attrs)$ to $\mathcal{H}_j$ over $\mathcal{F}_{\mathsf{smt}}$.
2. $\mathcal{H}_j$ upon receiving $(sid, jsid, A, e, s, attrs)$ from $\mathcal{I}$ verifies and stores the credential.
   - Compute $b \leftarrow g_1 \cdot h_0^s \cdot Q \cdot \prod_{i=1}^{L} h_{i+1}^{a_i}$.
   - Check that $A \neq 1_{\mathbb{G}_1}$ and $e(A, wg_2^e) = e(b, g_2)$.
   - Store $(sid, \mathcal{M}_i, (A, e, s), b, attrs)$ and output $(\mathsf{JOINED}, sid, jsid)$.

`Sign Request.` The sign protocol runs between a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$. After joining, together they can sign a message m with respect to a basename bsn, attribute predicate (D, I), and signature-based revocation list `SRL`. Again, we use a unique sub-session identifier $ssid$ to allow for multiple sign sessions.

1. $\mathcal{H}_j$ upon input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, (D, I), \mathtt{SRL})$ checks whether his attributes fulfill the predicate and randomizes the BBS+ credential:
   - Retrieve the join record $(sid, \mathcal{M}_i, (A, e, s), attrs)$.
   - Check that the attributes fulfill the predicate: Parse I as $(a'_1, \ldots, a'_L)$ and $attrs$ as $(a_1, \ldots, a_L)$ and check that $a_i = a'_i$ for every $i \in D$.
   - Send $(sid, ssid, m, bsn, (D, I), \mathtt{SRL})$ to $\mathcal{M}_i$ and store $(sid, ssid, a)$.
2. $\mathcal{M}_i$ upon receiving $(sid, ssid, m, bsn, (D, I), \mathtt{SRL})$ from $\mathcal{H}_j$ asks for permission to proceed.
   - Check that a join record $(sid, \mathcal{H}_j, gsk)$ exists.

– Store $(sid, ssid, m, bsn, (D, I), \texttt{SRL})$ and output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, (D, I), \texttt{SRL})$.

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

1. $\mathcal{M}_i$ upon input $(\mathsf{SIGNPROCEED}, sid, ssid)$ computes the pseudonym nym and starts proving knowledge of the credential.
   – Retrieve join record $(sid, \mathcal{H}_j, gsk)$ and sign record $(sid, ssid, m, bsn, (D, I), \texttt{SRL})$.
   – Set $nym \leftarrow \mathsf{H}_1(bsn)^{gsk}$.
   – Take $r_{gsk} \xleftarrow{\$} \mathbb{Z}_p$ and compute $E \leftarrow h_1^{r_{gsk}}$ and $L \leftarrow \mathsf{H}_1(bsn)^{r_{gsk}}$.
   – Send $(sid, ssid, E, L, nym)$ to $\mathcal{H}_j$.
2. $\mathcal{H}_j$ upon receiving $(sid, ssid, E, L, nym)$ from $\mathcal{M}_i$, randomizes the credential and completes the commitment phase of the zero-knowledge proof.
   – Choose $r_1 \xleftarrow{\$} \mathbb{Z}_p^*$ and set $A' \leftarrow A^{r_1}$. Set $r_3 \leftarrow \frac{1}{r_1}$.
   – Set $\bar{A} \leftarrow A'^{-e} \cdot b^{r_1} (= A'^{x})$.
   – Choose $r_2 \xleftarrow{\$} \mathbb{Z}_p$, set $d \leftarrow b^{r_1} \cdot h_0^{-r_2}$, and set $s' \leftarrow s - r_2 \cdot r_3$.
   – Using the TPM's contribution, the host computes our proof of knowledge of a BBS+ signature, as introduced in Section 4, extended with a proof that the pseudonym is correctly formed:

$$\pi \leftarrow SPK\{(gsk, \{a_i\}_{i \in \bar{D}}, e, r_2, r_3, s') : \bar{A}/d = A'^{-e} \cdot h_0^{r_2} \ \wedge \ g_1 \prod_{i \in D} {h_{i+1}}^{a_i} = d^{r_3} h_0^{-s'} h_1^{-gsk} \prod_{i \in \bar{D}} {h_{i+1}}^{-a_i} \ \wedge \ nym = \mathsf{H}_1(bsn)^{gsk}\}(m)$$

   – Take $r_{a_i} \xleftarrow{\$} \mathbb{Z}_p$ for $i \in \bar{D}$, and $r_e, r_{r_2}, r_{r_3}, r_{s'} \xleftarrow{\$} \mathbb{Z}_p$.
   – Compute t-values

$$t_1 \leftarrow A'^{r_e} \cdot h_0^{r_{r_2}}$$
$$t_2 \leftarrow d^{r_{r_3}} h_0^{r_s} E^{-1} \prod_{i \in \bar{D}} {h_{i+1}}^{r_{a_i}}$$

   – Compute $c' \leftarrow \mathsf{H}(A', \bar{A}, d, nym, t_1, t_2, L, g_1, h_0, \ldots, h_L, w)$.
   – Send $(sid, ssid, c')$ to $\mathcal{M}_i$.
3. $\mathcal{M}_i$ upon receiving $(sid, ssid, c')$ from $\mathcal{H}_j$.
   – Take a nonce $n \xleftarrow{\$} \{0,1\}^{\tau}$.
   – Compute $c \leftarrow \mathsf{H}(n, c', m, bsn, (D, I), \texttt{SRL})$.
   – Set $s_{gsk} \leftarrow r_{gsk} + c \cdot gsk$.
   – Send $(sid, ssid, s_{gsk})$ to $\mathcal{H}_j$.
4. $\mathcal{H}_j$ upon receiving $(sid, ssid, s_{gsk})$ from $\mathcal{M}_i$, completes the zero-knowledge proof.

– Set $s_{a_i} \leftarrow r_{a_i} - c \cdot a_i$ for $i \in \bar{D}$, $s_e \leftarrow r_e - c \cdot e$, $s_{r_2} \leftarrow r_{r_2} + c \cdot r_2$, $s_{r_3} \leftarrow r_{r_3} + c \cdot r_3$, $s_s \leftarrow r_s - c \cdot s'$.
– Set $\pi \leftarrow (c, s_{gsk}, \{s_{a_i}\}_{i \in \bar{D}}, s_e, s_{r_2}, s_{r_3}, s_s, n)$.

5. As signature-based revocation is used, a revocation list SRL containing tuples (bsn_i, nym_i) is given, and the platform must prove that $\mathsf{H}_1(bsn_i)^{gsk} \neq nym_i$. It does so using the Camenisch-Shoup proof of inequality of discrete logarithms [CS03]: take a random γ, compute $C \leftarrow (\mathsf{H}_1(bsn_i)^{gsk}/nym_i)^{\gamma}$, and prove $SPK\{(\alpha, \beta) : C = \mathsf{H}_1(bsn_i)^{\alpha}(\frac{1}{nym_i})^{\beta} \wedge 1 = \mathsf{H}_1(bsn)^{\alpha}(\frac{1}{nym})^{\beta}\}$. For every $(bsn_i, nym_i) \in \mathtt{SRL}$, the platform takes the following steps.
   (a) Host $\mathcal{H}_j$ sends $(sid, ssid, bsn_i)$ to $\mathcal{M}_i$.
   (b) Upon receiving $(sid, ssid, bsn_i)$, the TPM $\mathcal{M}_i$ starts the commitment phase of this proof of non-revocation.
      – Take $r_{i,\alpha} \xleftarrow{\$} \mathbb{Z}_p$ and compute $t'_{i,1} \leftarrow \mathsf{H}_1(bsn_i)^{r_{i,\alpha}}$, $t'_{i,2} \leftarrow \mathsf{H}_1(bsn)^{r_{i,\alpha}}$, $K \leftarrow \mathsf{H}_1(bsn_i)^{gsk}$.
      – Send $(sid, ssid, t'_{i,1}), t'_{i,2}, K)$ to $\mathcal{H}_j$.
   (c) Upon receiving $(sid, ssid, t'_{i,1}), t'_{i,2}, K)$, $\mathcal{H}_j$ completes the commitment phase of the non-revocation proof.
      – Take $\gamma_i \xleftarrow{\$} \mathbb{Z}_p$ and set $C_i \leftarrow (K/nym_i)^{\gamma_i}$.
      – Check $C_i \neq 1_{\mathbb{G}_1}$.
      – Take $r_{i,\beta} \xleftarrow{\$} \mathbb{Z}_p$ and set $t_{i,1} \leftarrow t'^{i,\beta}_{i,1} \cdot (\frac{1}{nym_i})^{r_{i,\beta}}$ and $t_{i,2} \leftarrow t'^{\gamma_i}_{i,2} \cdot (\frac{1}{nym})^{r_{i,\beta}}$.
      – Compute $c' \leftarrow \mathsf{H}(C, bsn_i, bsn, nym_i, nym, n, t_{i,1}, t_{i,2})$
      – Send $(sid, ssid, c')$ to $\mathcal{M}_i$.
   (d) $\mathcal{M}_i$ upon receiving $(sid, ssid, c')$ from $\mathcal{H}_j$.
      – Take nonce $n_i \xleftarrow{\$} \{0,1\}^{\tau}$ and compute $c \leftarrow \mathsf{H}(n_i, c)$.
      – Set $s'_{i,\alpha} \leftarrow r_{i,\alpha} + c \cdot gsk$ and send $(sid, ssid, s'_{i,\alpha}, n_i)$ to $\mathcal{H}_j$.
   (e) Upon receiving $(sid, ssid, s'_{i,\alpha}, n_i)$ from $\mathcal{M}_i$, host $\mathcal{H}_j$ finishes the non-revocation proof.
      – Compute $c \leftarrow \mathsf{H}(n_i, c')$.
      – Set $s_{i,\alpha} \leftarrow \gamma \cdot s'_{i,gsk}$ and $s_{i,\beta} \leftarrow r_{i,\beta} + c \cdot \gamma$.
      – Set $\pi_i \leftarrow (c, n_i, C_i, s_{i,\alpha}, s_{i,\beta})$.
6. The host outputs $(\mathsf{SIGNATURE}, sid, ssid, (A', \bar{A}, d, nym, \pi, \{\pi_i\}))$.

`Verify.` The verify algorithm allows one to check whether a signature σ on message m with respect to basename bsn, attribute disclosure (D, I), private key revocation list RL, and signature revocation list SRL is valid.

1. $\mathcal{V}$ upon input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, (D, I), \mathtt{RL}, \mathtt{SRL})$ verifies the signature:
   – Parse σ as $(A', \bar{A}, d, nym, \pi, \{\pi_i\})$.
   – Check $A' \neq 1_{\mathbb{G}_1}$ and $e(A', X) = e(\bar{A}, g_2)$.
   – Verify π with respect to A', $\bar{A}$, d, and nym:
      • Parse π as $(c, s_{gsk}, \{s_{a_i}\}_{i \in \bar{D}}, s_e, s_{r_2}, s_{r_3}, s_f, n)$.

- Set:

$$\hat{L} \leftarrow h_1^{s_{gsk}} \cdot nym^{-c}$$
$$\hat{t}_1 \leftarrow A'^{s_e} \cdot h_0^{s_{r_2}} \cdot (\bar{A}/d)^{-c}$$
$$\hat{t}_2 \leftarrow d^{s_{r_3}} h_0^{s_f} h_1^{-s_{gsk}} \prod_{i \in \bar{D}} {h_{i+1}}^{-s_{a_i}} \cdot (g_1 \prod_{i \in D} {h_{i+1}}^{a_i})^{-c}$$

  - Check $c = \mathsf{H}(n, \mathsf{H}(A', \bar{A}, d, nym, \hat{t}_1, \hat{t}_2, \hat{L}, g_1, h_0, \ldots, h_L, w), m, bsn, (D, I), \mathtt{SRL})$.

– For every $gsk_i \in \mathtt{RL}$, check that $\mathsf{H}_1(bsn)^{gsk_i} \neq nym$.
– For every $(bsn_i, \pi_i) \in \mathtt{SRL}$:
  - Parse π_i as $(c, n_i, C_i, s_{i,\alpha}, s_{i,\beta})$.
  - Check $C \neq 1_{\mathbb{G}_1}$.
  - Set $\hat{t}_{i,1} \leftarrow \mathsf{H}_1(bsn_i)^{s_{i,\alpha}} \frac{1}{nym_i}^{s_{i,\beta}}$ and $\hat{t}_{i,2} \leftarrow \mathsf{H}_1(bsn)^{s_{i,\alpha}} \frac{1}{nym}^{s_{i,\beta}}$.
  - Check $c = \mathsf{H}(n_i, \mathsf{H}(C, bsn_i, bsn, nym_i, nym, n, \hat{t}_{i,1}, \hat{t}_{i,2}))$.
– If all tests pass, set $f \leftarrow 1$, otherwise $f \leftarrow 0$.
– Output $(\mathsf{VERIFIED}, sid, f)$.

`Link.` The link algorithm allows one to check whether two signatures σ, σ', on messages m, m' respectively, that were generated for the same basename bsn were created by the same TPM.

1. $\mathcal{V}$ upon input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ verifies the signatures and compares the pseudonyms contained in σ, σ':
   – Check that both signatures σ, σ' are valid with respect to $m, bsn, p, \mathtt{SRL}$ and $m', bsn, p', \mathtt{SRL}'$ respectively. Output $\perp$ if they are not both valid.
   – Parse the signatures as $(A', \bar{A}, d, nym, \pi, \{\pi_i\}) \leftarrow \sigma$ and $(A'', \bar{A}', d', nym', \pi', \{\pi_i'\}) \leftarrow \sigma'$.
   – If $nym = nym'$, set $f \leftarrow 1$, otherwise $f \leftarrow 0$.
   – Output (LINK, sid, f).

5.2 Comparison with Previous DAA Schemes

Our protocol is quite similar to the most recent qSDH-based DAA schemes [Che09,BL10,CU15]. However, a few key changes were needed to achieve provable security and address the problems mentioned in Sect. 2. First, we use a BBS+ signature for the membership credential, instead of the simplified credential where the s-value is ommited as used in the recent schemes [Che09, BL10, CU15]. This credential is proven to be unforgeable, where the simplified version is not.

In addition, we change the way the platform proves knowledge of a membership credential. Our proof is more efficient to generate, as it works completely in $\mathbb{G}_1$, avoiding the less efficient groups $\mathbb{G}_2$, $\mathbb{G}_T$ and the pairing operations. Verification requires only two pairing operations, the rest of the computation takes place in $\mathbb{G}_1$.

	$\mathcal{M}$ Sign	$\mathcal{H}$ Sign	Est. $\mathcal{H}$ Sign time (ms)	Verify	Est. ver. time (ms)
[CF08]	$2\mathbb{G}_1, 1\mathbb{G}_T$	$1\mathbb{G}_1, 2\mathbb{G}_1^2, 1\mathbb{G}_T, 1P$	50.1	$1\mathbb{G}_1^2, 2\mathbb{G}_1^3, 1\mathbb{G}_T^5, 3P$	172.7
[Che09]	$2\mathbb{G}_1, 1\mathbb{G}_T$	$1\mathbb{G}_1, 1\mathbb{G}_T^3$	58.9	$1\mathbb{G}_1^2, 1\mathbb{G}_2^2, 1\mathbb{G}_T^4, 1P$	111.0
[BL10]	$3\mathbb{G}_1$	$1\mathbb{G}_1, 1\mathbb{G}_1^2, 1\mathbb{G}_T, 1P$	45.7	$1\mathbb{G}_1^2, 1\mathbb{G}_2^2, 1\mathbb{G}_T^4, 1P$	111.0
[CPS10]	$3\mathbb{G}_1$	$4\mathbb{G}_1$	8.8	$2\mathbb{G}_1^2, 4P$	89.6
[CU15]-1	$3\mathbb{G}_1$	$(4+L+U)\mathbb{G}_1$	8.8	$2\mathbb{G}_1, 2\mathbb{G}_1^L, 2\mathbb{G}_1^D, 2\mathbb{G}_1^U, 6P$	125.6
[CU15]-2	$3\mathbb{G}_1$	$2\mathbb{G}_1, 1\mathbb{G}_1^{U+2}, 2P$	49.2	$1\mathbb{G}_1^2, 1\mathbb{G}_1^{4+L}, 2P$	53.6
[CDL16b]	$5\mathbb{G}_1$	$4\mathbb{G}_1$	8.8	$2\mathbb{G}_1^2, 4P$	89.6
This work[4]	$3\mathbb{G}_1$	$1\mathbb{G}_1, 1\mathbb{G}_1^{2+U}, 1\mathbb{G}_T, 1P$	45.7	$1\mathbb{G}_1^2, 1\mathbb{G}_2^2, 1\mathbb{G}_T^{4+L}, 1P$	111.0
This work	$3\mathbb{G}_1$	$2\mathbb{G}_1, 2\mathbb{G}_1^2, 1\mathbb{G}_1^{2+U}$	17.6	$1\mathbb{G}_1^2, 1\mathbb{G}_1^3, 1\mathbb{G}_1^{5+L}, 2P$	60.2

Table 1. A comparison of the efficiency of DAA schemes.

Compared to the most recent EPID scheme by Brickell and Li [BL11], we introduce a way to split the workload between a TPM and host, and add basenames steering linkability. The usage of basenames is required to prevent the TPM from serving as a static Diffie-Hellman oracle towards the host. For non-revocation proofs, the platform must prove that its pseudonym $nym = B^{gsk}$ is based on a different key than a pseudonym in a revoked signature $nym' = B'^{gsk'}$. A host proving the inequality of the keys with the help of a TPM using the method by Camenisch and Shoup will learn B'^{gsk}, for any B' of its choosing. By requiring basenames, i.e., $B = \mathsf{H}_1(bsn)$, learning $B'^{gsk} = \mathsf{H}_1(bsn)^{gsk}$ does not give a corrupt host any information, as in the random oracle model this can be simulated without knowing gsk.

For the reason mentioned above, the fully anonymous option $bsn = \perp$ from previous DAA schemes is not supported by our scheme, but we argue that this does not affect privacy: A platform can choose a fresh basename it only uses once to be fully anonymous. Any verifier that accepts fully anonymous signatures can simply accept signatures with respect to any basename.

Compared to the existing DAA-A scheme [CU15], we store all attributes except the secret key on the host for efficiency. This still guarantees unforgeability with an honest TPM and corrupt host. Anonymity is not affected either, as in either case, the host must be trusted for anonymity.

In Table 1 we compare the computational efficiency of our scheme with the other DAA schemes. In particular, we show the computational cost for the TPM in the sign algorithm, for the host in the sign algorithm, and for the verifier in the verify algorithm, as these are the algorithms that will be used frequently. We denote k exponentiations in group $\mathbb{G}_i$ by $k\mathbb{G}_i$, $k\mathbb{G}_i^j$ denotes k j-multi-exponentiations, and kP denotes k pairing operations. Estimated running time of the algorithms (without using attributes) are computed by analyzing

[4] We updated our paper with improved results since the conference version. This row contains the original results.

	Cred. Size	Cred. Size (bytes)	Signature Size	Sig. Size (bytes)
[CF08]	$2\mathbb{Z}_p$ $1\mathbb{G}_1$	97	$6\mathbb{Z}_p$ $2\mathbb{G}_1$ $2\mathbb{G}_T$ $1H$	1058
[Che09]	$1\mathbb{Z}_p$ $1\mathbb{G}_1$	65	$4\mathbb{Z}_p$ $3\mathbb{G}_1$ $1H$	259
[BL10]	$1\mathbb{Z}_p$ $1\mathbb{G}_1$	65	$4\mathbb{Z}_p$ $3\mathbb{G}_1$ $1H$	259
[CPS10]	$4\mathbb{G}_1$	132	$1\mathbb{Z}_p$ $6\mathbb{G}_1$ $1H$	262
[CU15]-1	$(5+L)\mathbb{G}_1$	165	$(1+U)\mathbb{Z}_p$ $(7+L)\mathbb{G}_1$ $1H$	295
[CU15]-2	$1\mathbb{Z}_p$ $1\mathbb{G}_1$	65	$(5+U)\mathbb{Z}_p$ $3\mathbb{G}_1$ $1H$	259
[CDL16b]	$4\mathbb{G}_1$	132	$1\mathbb{Z}_p$ $6\mathbb{G}_1$ $1H$	229
This work[4]	$2\mathbb{Z}_p$ $1\mathbb{G}_1$	97	$(5+U)\mathbb{Z}_p$ $2\mathbb{G}_1$ $1H$	258
This work	$2\mathbb{Z}_p$ $1\mathbb{G}_1$	97	$(5+U)\mathbb{Z}_p$ $4\mathbb{G}_1$ $1H$	324

Table 2. A comparison of the credential and signature size of DAA schemes.

the running time of the primitive operations[5] on the 256-bit BN curve [BN06] supported by the TPM2.0.

Table 2 gives a comparison of the size of credentials and of signatures with other DAA schemes. Here, $k\mathbb{G}$ denotes the bits required to represent k elements of $\mathbb{G}$, and H denotes the bit length of the hash output. The byte size value is computed using the 256-bit BN curve [BN06] supported by TPM2.0 using point compression. This column does not consider attributes to allow comparison with schemes that lack support for attributes. CU15-1 denotes the LRSW-based DAA-A scheme by Chen and Urian [CU15], and CU15-2 the qSDH-based instantiation. We analyzed both schemes for signatures with only the secret key on the TPM, which is used to create a pseudonym, and all other attributes held by the host. We let L denote the amount of attributes, with D the amount of disclosed attributes and U the amount of undisclosed attributes. Revocation lists and revocation checks are omitted for these efficiency numbers.

To compare this scheme with previous DAA schemes, we consider the efficiency without attributes, i.e., $L = D = U = 0$. Our TPM signing operation is as efficient as the most efficient DAA schemes, and only works in $\mathbb{G}_1$. LRSW-based schemes [CPS10, CDL16b] have a more efficient signing operation for the host, but our scheme is much more efficient than existing qSDH-based schemes. The verification time of our scheme beats the LRSW-based schemes and is almost as fast as the fastest verification operation. The credential size slightly increases to allow for our improved efficiency.

We stress that many of the listed schemes are not provably secure, whereas we rigorously prove our scheme secure.

[5] Analyzed by executing the operations 100000 times and averaging, using the IAIK ECCelerate library (`http://jcewww.iaik.tu-graz.ac.at/`), using a single thread of an Intel Core i5 4300u CPU. We did not use multiexponentiations.

6 Security Analysis

Theorem 1. *The protocol $\Pi_{\mathsf{daa}+}$ presented in Section 5 securely realizes $\mathcal{F}^{l}_{\mathsf{daa}+}$ in the $(\mathcal{F}_{\mathsf{auth}*}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}^{l}_{\mathsf{smt}}, \mathcal{F}^{D}_{\mathsf{crs}})$-hybrid model using random oracles and static corruptions, if the DDH and the JOC version of the qSDH assumptions hold, and the proofs-of-knowledge are online extractable.*

Proof. We have to prove that our scheme realizes $\mathcal{F}^{l}_{\mathsf{daa}+}$, which means proving that for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for every environment $\mathcal{E}$ we have $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{E}} \approx \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{E}}$.

To show that no environment $\mathcal{E}$ can distinguish the real world, in which it is working with $\Pi_{\mathsf{daa}+}$ and adversary $\mathcal{A}$, from the ideal world, in which it uses $\mathcal{F}^{l}_{\mathsf{daa}+}$ with simulator $\mathcal{S}$, we use a sequence of games. We start with the real world protocol execution. In the next game we construct one entity $\mathcal{C}$ that runs the real world protocol for all honest parties. Then we split $\mathcal{C}$ into two pieces, 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, and gradually change $\mathcal{F}$ and update $\mathcal{S}$ accordingly, to end up with the full $\mathcal{F}^{l}_{\mathsf{daa}+}$ and a satisfying simulator. All intermediate functionalities and simulators are given in Appendix A. We now show that every game hop is indistinguishable from the previous. Note that although we separate $\mathcal{F}$ and $\mathcal{S}$, in reductions we can consider them to be one, as this does not affect $\mathcal{A}$ and $\mathcal{E}$.

Game 1: This is the real world protocol.

Game 2: $\mathcal{C}$ receives all inputs for honest parties and simulates the real world protocol for honest parties. By construction, this is equivalent to the real world.

Game 3: We now split $\mathcal{C}$ into two pieces, $\mathcal{F}$ and $\mathcal{S}$. We let $\mathcal{F}$ evolve to become $\mathcal{F}^{l}_{\mathsf{daa}+}$ and $\mathcal{S}$ to become the simulator. $\mathcal{F}$ behaves as an ideal functionality, so the messages it sends are authenticated and immediate, $\mathcal{A}$ will not notice them. $\mathcal{F}$ receives all the inputs, who forwards them to $\mathcal{S}$. $\mathcal{S}$ will simulate the real world protocol for all honest parties, and sends the outputs to $\mathcal{F}$, who forwards them to $\mathcal{E}$. Outputs generated by parties simulated by $\mathcal{S}$ are not sent anywhere, only $\mathcal{S}$ notices them. $\mathcal{S}$ sends an equivalent output to $\mathcal{F}$ using an OUTPUT message, such that $\mathcal{F}$ will generate the same output.

This game is simply Game 2 but structured differently, so Game 3 = Game 2.

Game 4: We now change the behavior of $\mathcal{F}$ in the setup interface, and store algorithms in $\mathcal{F}$. Note that $\mathcal{F}$ now checks the structure of *sid* for honest issuer $\mathcal{I}$, and aborts when it is not of the expected form. This abort will not change the view of $\mathcal{E}$, as $\mathcal{I}$ performs the same check upon receiving this input.

The simulated real world does not change, as $\mathcal{I}$ gives "$\mathcal{I}$" the correct input. For corrupt $\mathcal{I}$, $\mathcal{S}$ also extracts the secret key and calls the setup interface on $\mathcal{I}$'s behalf, but clearly this does not change $\mathcal{E}$'s view, so Game 4 = Game 3.

Game 5: $\mathcal{F}$ now handles verification and link queries instead of forwarding them to $\mathcal{S}$. There are no protocol messages, so we only have to make sure the output is equal.

The verification algorithm $\mathcal{F}$ uses is almost equal to the real world protocol. The only difference is the that the ver algorithm that $\mathcal{F}$ uses does not contain the private-key revocation check. $\mathcal{F}$ performs this check separately, so the outcomes are equal.

The linking protocol outputs $\perp$ when it is called with invalid signatures. $\mathcal{F}$ does the same, it verifies the signatures and outputs $\perp$ when one of the signatures is not valid. The protocol then checks the pseudonyms for equality, which is exactly what $\mathcal{F}$ does, showing that the outputs will be equal. $\mathcal{F}$ requires link to be symmetrical and outputs $\perp$ when it notices that it is not. This algorithm is symmetrical, so this abort will not happen and we have Game 5 = Game 4.

Game 6: The join interface of $\mathcal{F}$ is now changed. It stores which members joined, and if $\mathcal{I}$ is honest, stores the key gsk with which corrupt TPMs joined. $\mathcal{S}$ extracts this from proof π in the simulated real world.

If extraction by rewinding is used, a fresh nonce must be included in the proof, guaranteeing that this proof is fresh and we can extract from it. Note that we can rewind here as the honest $\mathcal{I}$ only allows a logarithmic number of simultaneous join sessions, so there is no risk of requiring exponential time to rewind every join session.

$\mathcal{S}$ always has enough information to simulate the real world protocol, except when only the issuer is honest. It then does not know which host initiated this join, so it cannot make a join query with $\mathcal{F}$ on that host's behalf. However, it is sufficient to take any corrupt host, as this will only result in a different identity in $\mathtt{Members}$, and in $\mathcal{F}^{l}_{\mathsf{daa}+}$ this identity only matters for honest hosts.

We must argue that $\mathcal{F}$ does not prevent an execution that was allowed in the previous game. $\mathcal{F}$ only has one check that may cause it to abort, if $\mathcal{M}$ already registered and $\mathcal{I}$ is honest. Since in the protocol, $\mathcal{I}$ checks this before outputting $\mathsf{JOINPROCEED}$, $\mathcal{F}$ will never abort.

As $\mathcal{S}$ can simulate the real world protocol and keep everything in sync with $\mathcal{F}$, the view of $\mathcal{E}$ does not change, Game 6 = Game 5.

Game 7: $\mathcal{F}$ now creates pseudonymous signatures for honest platforms using the algorithms defined in setup. One difference is that the signature made by $\mathcal{F}$ will use a credential that contains dummy attribute values for the nondisclosed attributes. This change is not noticable, as only $(A', \bar{A}, d)$ and proof π are affected. These values will have the same distribution when dummy attribute values are used, A' and d are still independantly at random in $\mathbb{G}_1$, and π is indistinguishable by the zero knowledge property. Another change is that signatures will now be based on fresh gsk values for every basename, whereas before all signatures were based on the same gsk. This changes signature-based revocation, as it proves that the gsk used differs from keys used in revoked signatures, and the pseudonym values. Since $\mathcal{F}$ already checks that the platform is not revoked by the signature-based revocation list $\mathtt{SRL}$, sig will still succeed in creating a

valid proofs of non-revocation $\{\pi_i\}$. To show that $\mathcal{E}$ cannot notice the difference in the computation of the pseudonyms, we make this change gradually.

In GAME $7.k.k'$, $\mathcal{F}$ forwards all signing inputs with $\mathcal{M}_i, i > k$ to $\mathcal{S}$, and $\mathcal{S}$ creates signatures as before. Inputs with $\mathcal{M}_i, i < k$ are handled by $\mathcal{F}$ using the algorithm. For signing inputs with $\mathcal{M}_k$, signing queries with the first k' basenames that are hashed are handled by $\mathcal{F}$, and later inputs will be forwarded to $\mathcal{S}$. We now have GAME $7.0.0$ = GAME 6. When increasing k', in polynomially many steps GAME $7.k$,k' will be equal to GAME $7.k + 1.0$, as there can only be polynomially many basenames hashed. Repeating this process will make k large enough to include all TPMs, so for some k, k', we have GAME 7 = GAME $7.k.k'$. Therefore, to show that GAME 7 = GAME 6, it suffices to show that increasing k' by one is indistinguishable.

We now show that anyone distinguishing GAME $7.k.k'$ from GAME $7.k.k'+1$ can solve DDH. We modify $\mathcal{S}$ working with $\mathcal{F}$ parametrized by k, k' such that if it receives a DDH tuple, it is equivalent to GAME $7.k.k'$, and otherwise equivalent to GAME $7.k.k' + 1$.

$\mathcal{S}$ receives a DDH instance $\tilde{g}, \alpha, \beta, \gamma \in \mathbb{G}_1$ and must answer whether $log_{\tilde{g}}(\alpha) \cdot log_{\tilde{g}}(\beta) = log_{\tilde{g}}(\gamma)$. $\mathcal{S}$ sets $h_1 \leftarrow \tilde{g}$, which it can do as it simulates $\mathcal{F}_{\mathsf{crs}}$. $\mathcal{S}$ answers H_1 queries on basenames bsn_i with $h_1^{r_i}$ for some $r_i \xleftarrow{\$} \mathbb{Z}_p$, maintaining consistency, except the k'-th query, in which it returns β. $\mathcal{S}$ simulates $\mathcal{M}_k$ using the unknown discrete logarithm of α as its gsk by setting $Q \leftarrow \alpha$, simulating proof π_1 in join. Signing queries are handled as follows. When $\mathcal{F}$ signs on behalf of $\mathcal{M}_i$ with $i < k$, the query is forwarded to $\mathcal{S}$ who handles it as before using the real world protocol. If $i > k$, $\mathcal{F}$ handles the signing using the algorithms supplied by $\mathcal{S}$ and fresh gsk values per basename. When $\mathcal{F}$ signs on behalf of $\mathcal{M}_k$ with basename bsn_i and $i < k'$, it sets $nym \leftarrow \alpha^{r_i}$. If $i = k'$, $\mathcal{S}$ sets $nym \leftarrow \gamma$. If $i > k'$, $\mathcal{S}$ takes $gsk_i \xleftarrow{\$} \mathbb{Z}_p$ and sets $nym \leftarrow \mathsf{H}_1(bsn_i)^{gsk_i}$, reusing this gsk_i for further signing queries of $\mathcal{M}_k$ with bsn_i. In either case, $\mathcal{M}_{k'}$ simulates $(A', \bar{A}, d, \pi)$ as shown in Lemma 2.

When the DDH instance is a DDH tuple, this game is GAME $7.k.k'$, and when it is not, this game is GAME $7.k.k'+1$. Therefore, any distinguisher distinguishing these two games can break DDH, so GAME 7 $\approx$ GAME 6.

Game 8: $\mathcal{F}$ no longer leaks $m, bsn, p, \mathtt{SRL}$ to $\mathcal{S}$, but only the leakage $l(m, bsn, p, \mathtt{SRL})$. All the adversary notices is the leakage of the secure channel between the TPM and host. $\mathcal{S}$ can still simulate this by taking dummy values that result in the same leakage. As $\mathcal{S}$ makes sure the dummy attribute predicate still holds and the signature revocation list does not revoke the platform, any signing query that would previously succeed will still succeed, so GAME 8 $\approx$ GAME 7.

Game 9: $\mathcal{F}$ now only allows platforms that joined with attributes fulfilling the attribute predicate to sign when $\mathcal{I}$ is honest. This check will not change the view of $\mathcal{E}$. Before signing with some $\mathcal{M}_i$ in the real world, an honest host will check whether it joined with $\mathcal{M}_i$ and abort otherwise, so for honest hosts there is no difference. An honest TPM $\mathcal{M}_i$ only signs when it has joined with that host, and when an honest $\mathcal{M}_i$ performs the join protocol with a corrupt $\mathcal{H}_j$ and honest $\mathcal{I}$,

the simulator will make a join query with $\mathcal{F}$, ensuring that $\mathcal{M}_i$ and $\mathcal{H}_j$ are in `Members`. Since $\mathcal{F}$ still allows any signing that could take place in the real world, Game 9 = Game 8.

Game 10: $\mathcal{F}$ no longer informs the adversary of the attributes of an honest platform in the join protocol. $\mathcal{S}$ now uses dummy attributes in the join protocol, but as all messages are sent over secure channels, this does not change the adversary's view.

In the real world sign protocol, the host aborts immediately when it is given a attribute predicate that does not hold for its attributes. As we do not know the real attributes, we cannot simulate this. However, the first check $\mathcal{F}$ makes checks whether the attribute predicate holds for the attributes of the platform, and only then will $\mathcal{S}$ be notified. This way, $\mathcal{S}$ knows that it has to simulate with some attribute predicate that holds for the dummy attributes. The signature will be generated by $\mathcal{F}$, and will therefore contain the correct attributes and attribute predicate.

Game 11: When storing a new gsk, $\mathcal{F}$ checks $\mathsf{CheckGskCorrupt}(gsk) = 1$ or $\mathsf{CheckGskHonest}(gsk) = 1$. We now show that these checks will never fail.

Note that we only consider valid signatures from `VerResults`, and `Signed` only contains valid signatures (added for honest TPM and host) and $\perp$ (added for honest TPM with corrupt host). As $\mathsf{identify}(\perp) = 0$, we only have to consider valid signatures.

Any signature that passes verification has $nym = \mathsf{H}_1(bsn)^{gsk}$. With overwhelming probability, $\mathsf{H}_1(bsn) \neq 1$, meaning it generates $\mathbb{G}_1$ and there is only one $gsk \in \mathbb{Z}_p$ that has $nym = \mathsf{H}_1(bsn)^{gsk}$. From this property, it follows that $\mathsf{CheckGskCorrupt}$ can never fail.

The gsk values that enter $\mathsf{CheckGskHonest}$ are taken uniformly at random from $\mathbb{Z}_p$, which has exponential size, meaning that the probability that one of the existing signatures contains that gsk is negligible. Therefore we have Game 11 $\approx$ Game 10.

Game 12: $\mathcal{F}$ now performs some checks on honestly generated signatures. First, it checks that these signatures verify. This check will always pass, as sig creates valid signatures.

Second, it makes sure $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$. $\mathcal{F}$ running sig sets nym such that $nym = \mathsf{H}_1(bsn)^{gsk}$, so $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$.

Third, it checks that no honest user is already using gsk. We reduce this check happening with non-negligible probability to solving the Discrete Logarithm (DL) problem.[6] $\mathcal{F}$ receives an instance $\alpha \in \mathbb{G}_1$ of the DL problem and must answer $log_{h_1}(\alpha)$. Only polynomially many gsk values are created in signing requests, $\mathcal{F}$ chooses one of those at random. Instead of setting $gsk \leftarrow \mathsf{ukgen}$, $\mathcal{F}$ creates a credential on α, determines the nym using the power $\mathcal{S}$ has over random oracle, and simulates the π. When $\mathcal{F}$ would reuse this key, it repeats the same process. When a key matching any of these signatures is found in `Members` or

[6] Note that the DL assumption is not mentioned in the theorem statement as it is implied by the qSDH assumption.

`DomainKeys`, this must be the discrete log of α, as there is only one *gsk* matching a signature (assuming $\mathsf{H}_1(bsn) \neq 1$).

As every check passes with overwhelming probability, we have Game 12 $\approx$ Game 11.

Game 13: In verification, $\mathcal{F}$ now checks whether it finds multiple distinct *gsk* values matching the purported signature. We now show that this check triggers with negligible probability.

A matching key means $nym = \mathsf{H}_1(bsn)^{gsk}$, which can only hold for multiple keys if $\mathsf{H}_1(bsn) = 1$, which happens with negligible probability. Therefore Game 13 $\approx$ Game 12.

Game 14: When $\mathcal{I}$ is honest, $\mathcal{F}$ now only accepts signatures on *gsk* and *attrs* values on which $\mathcal{I}$ issued a credential. Under the qSDH assumption, this check changes the verification outcome only with negligible probability.

$\mathcal{C}$ receives a qSDH instance and use the proof of Lemma 1 to get a BBS+ public key and simulate the BBS+ signing queries. Furthermore, part of the qSDH instance is (g_1, g_1^x). It takes $\rho \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\bar{g}_1 \leftarrow g_1^\rho$ and $\bar{g}_2 \leftarrow (g_1^x)^\rho$. $\mathcal{C}$ registers the public key with a simulated proof. When $\mathcal{I}$ must create a credential in the join protocol, it takes the extracted *gsk* and the attribute values *attrs* and sends it to the signing oracle to receive (A, e, s). When $\mathcal{F}$ must sign on behalf of an honest platform, it also uses the signing oracle to create the signature, but now it uses the *attrs* the platform joined with, instead of dummy values for undisclosed attribute values. This change is not noticable, as the proofs-of-knowledge will be indistinguishable.

When $\mathcal{F}$ sees a valid signature with respect to predicate p, but no matching *gsk* has been found for a TPM with attributes matching p, we extract a credential on *gsk*, *attrs* with $p(attrs) = 1$, which we can do by Lemma 2. $\mathcal{I}$ never issued such a credential, so $\mathcal{C}$ wins the unforgeability game. As the BBS+ signature is unforgeable under the JOC version of the qSDH assumption (by Lemma 1), we have Game 14 $\approx$ Game 13.

Game 15: $\mathcal{F}$ now prevents forging signatures using an honest TPM's *gsk*. We make this change gradually, and in Game $15.i.i'$, we do this check for the first i TPMs, and for $\mathcal{M}_i$ we do this for the i' first basenames that are hashed. We show that any environment able to distinguish Game $15.i.i'$ and Game $15.i.i' + 1$ can break the DL assumption.

$\mathcal{S}$ receives an DL instance α. When $\mathcal{M}_i$ signs with basename $bsn_{i'}$, instead of picking a fresh basename, $\mathcal{F}$ uses the unknown discrete logarithm of α as the TPMs *gsk*. The credential can be created without knowledge of the discrete logarithm of α, and by its power over the random oracle it knows some r such that $\mathsf{H}_1(bsn_{i'}) = h_1^r$, and can set the pseudonym as α^r. The proof π can be simulated.

In verification, $\mathcal{F}$ does not know the discrete logarithm of α, so it cannot directly check whether a signature matches this key. However, instead it can check nym^{1/r_i} for signatures with bsn_i and $\mathsf{H}_1(bsn_i) = h_1^{r_i}$.

When $\mathcal{F}$ finds a signature matching $\mathcal{M}_i$ valid with respect to $m, bsn, p, \mathtt{SRL}$, but $\mathcal{M}_i$ never signed this, we can rewind the proof to extract gsk, which must be the discrete logarithm of α. This breaks the DL assumption, so Game 15 $\approx$ Game 14.

Game 16: $\mathcal{F}$ now prevents honest TPMs from being revoked. Any environment that can put a gsk on the revocation list that matches an honest TPMs signature can break the DL assumption. Note that for honest TPMs, there are only pairs $(\mathcal{M}_i, gsk)$ in `DomainKeys` that have $gsk \neq \perp$.

If this check aborts for a pair found in `Members`, $\mathcal{E}$ can solve the DL problem. $\mathcal{S}$ receives an instance $\alpha \in \mathbb{G}_1$ of the DL problem and must answer $log_{h_1}(\alpha)$. $\mathcal{S}$ chooses an honest TPM at random and sets $Q \leftarrow \alpha$ with a simulated proof in the join protocol, When a gsk from `Members` matching one of this TPMs signatures is found in the revocation list this must be the discrete log of h, as there is only one gsk matching a signature.

$\mathcal{F}$ receives an instance $\alpha \in \mathbb{G}_1$ of the DL problem and must answer $log_{h_1}(\alpha)$. Whenever $\mathcal{F}$ would choose a new gsk value when signing for an honest platform, instead it takes $r \xleftarrow{\$} \mathbb{Z}_p$ and creates a signature with gsk equal to r times the discrete log of α. It can do this as described in Game 15. When a key matching any of these signatures is found in the revocation list this must be r times the discrete log of α. This breaks the DL assumption, so Game 16 $\approx$ Game 15.

Game 17: $\mathcal{F}$ now performs the signature revocation check. If it found a matching key gsk that also matches some revoked signature, the signature will be rejected. This cannot change the verification outcome, as the ver algorithm checks that the platform proved that its gsk is unequal to the key used in the revoked signatures. Therefore Game 17 = Game 16.

Game 18: $\mathcal{F}$ now puts requirements on the link algorithm. These requirements do not change the output.

With overwhelming probability, we have $\mathsf{H}_1(bsn) \neq 1$, so thre is one unique $gsk \in \mathbb{Z}_p$ with $\mathsf{identify}(\sigma, m, bsn, gsk)$. If one gsk matches one of the signatures but not the other, then by soundness of the proof, $nym \neq nym'$ and link would also output 0. If both signatures match some gsk, then by soundness of the proof, we have $nym = nym'$ and link would also output 1. Therefore we have Game 18 = Game 17.

The functionality in Game 18 is equal to $\mathcal{F}^l_{\mathsf{daa}+}$, completing our security proof. □

Extraction from Zero-Knowledge Proofs. Theorem 1 is about the setting where we use online extractable zero-knowledge proofs. Instead of relying on *online extractable* SPKs one could also use extraction by rewinding, which would yield a more efficient scheme. However, one needs to take special care that the rewinding does not require exponential time in the security proof. The only SPK we constantly have to extract from in our security proof is π_1 used in the join protocol. Thus, we can avoid the exponential blow-up by letting the issuer limit the number of simultaneous join sessions to be logarithmic in the security param-

eter. Since we keep the way in which the simulator extracts witnesses abstract in the proof of Theorem 1, the very same simulator proves the scheme with extraction by rewinding secure. Note though, that the UC framework does not allow rewinding at all, i.e., this only proves the instantiation using extraction by rewinding secure in a stand-alone fashion, but one cannot claim composability guarantees.

7 Conclusion

DAA is one of the most complex cryptographic protocols deployed in practice. It is implemented in multiple platforms for trusted computing, including the Trusted Computing Group's TPM and Intel's SGX. A number of functional extensions to DAA have been proposed, including signature-based revocation and embedding of attributes. However, as we have shown in this paper, the security models and security proofs of the proposed DAA schemes based on the qSDH assumptions are not satisfactory. This includes the extended DAA schemes and the standardized DAA schemes. Bleichenbacher's attack [Ble98] on PKCS#1 demonstrates the importance of rigorous security proofs, in particular for cryptographic standards. It remains as future work, to revisit the concerned standards to eliminate the schemes' flaws and ensure that they are provably secure.

As a first step towards this, we have in this paper proposed a new DAA scheme with support for attributes and signature-based revocation. Our scheme is one of the most efficient DAA schemes. While the existing schemes do not have valid security proofs, our scheme is proven secure in the model by Camenisch et al. [CDL16b], extended to support attributes and signature-based revocation. As a side result, we have proven the BBS+ signature scheme to be secure in type-3 pairing settings, meaning our scheme can be used with the most efficient pairing-friendly elliptic curve groups.

References

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 111–125. Springer, Heidelberg, September 2006.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Heidelberg, May 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, April 2008.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 04*, pages 132–145. ACM Press, October 2004.

[BCL08] Ernie Brickell, Liqun Chen, and Jiangtao Li. A new direct anonymous attestation scheme from bilinear maps. In Peter Lipp, Ahmad-Reza Sadeghi, and Klaus-Michael Koch, editors, *Trusted Computing - Challenges and Applications, First International Conference on Trusted Computing and Trust in Information Technologies, Trust 2008, Villach, Austria, March 11-12, 2008, Proceedings*, volume 4968 of *Lecture Notes in Computer Science*, pages 166–178. Springer, 2008.

[BCL09] Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. *Int. J. Inf. Sec.*, 8(5):315–330, 2009.

[BFG+13] David Bernhard, Georg Fuchsbauer, Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. Anonymous attestation with user-controlled linkability. *Int. J. Inf. Sec.*, 12(3):219–249, 2013.

[BL07] Ernie Brickell and Jiangtao Li. Enhanced privacy id: a direct anonymous attestation scheme with enhanced revocation capabilities. In Peng Ning and Ting Yu, editors, *Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007, Alexandria, VA, USA, October 29, 2007*, pages 21–30. ACM, 2007.

[BL10] Ernie Brickell and Jiangtao Li. A pairing-based DAA scheme further reducing TPM resources. In Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi, editors, *Trust and Trustworthy Computing, Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010. Proceedings*, volume 6101 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2010.

[BL11] Ernie Brickell and Jiangtao Li. Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. *IJIPSI*, 1(1):3–33, 2011.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 1–12. Springer, Heidelberg, August 1998.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016. `http://eprint.iacr.org/2016/086`.

[CDH16] Jan Camenisch, Manu Drijvers, and Jan Hajny. Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, WPES '16, pages 123–133, New York, NY, USA, 2016. ACM.

[CDL16a] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In Michael Franz and Panos Papadimitratos, editors, *Trust and Trustworthy Computing - 9th International Conference, TRUST 2016, Vienna, Austria, August 29-30, 2016, Proceedings*, volume 9824 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2016.

[CDL16b] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 234–264. Springer, Heidelberg, March 2016.

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

[Che09] Liqun Chen. A DAA scheme requiring less TPM resources. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Information Security and Cryptology - 5th International Conference, Inscrypt 2009, Beijing, China, December 12-15, 2009. Revised Selected Papers*, volume 6151 of *Lecture Notes in Computer Science*, pages 350–365. Springer, 2009.

[CKY09] Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the portability of generalized Schnorr proofs. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 425–442. Springer, Heidelberg, April 2009.

[CMS08] Liqun Chen, Paul Morrissey, and Nigel P. Smart. Pairings in trusted computing (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 1–17. Springer, Heidelberg, September 2008.

[CPS10] Liqun Chen, Dan Page, and Nigel P. Smart. On the design and implementation of an efficient DAA scheme. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 14-16, 2010. Proceedings*, volume 6035 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2010.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Heidelberg, August 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

[CU15] Liqun Chen and Rainer Urian. DAA-A: direct anonymous attestation with attributes. In Mauro Conti, Matthias Schunter, and Ioannis G. Askoxylakis, editors, *Trust and Trustworthy Computing - 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*, volume 9229 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2015.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel 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.

[Int15] International Organization for Standardization. ISO/IEC 11889: Information technology - Trusted platform module library, 2015.

[LRSW99] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *SAC 1999*, volume 1758 of *LNCS*, pages 184–199. Springer, Heidelberg, August 1999.

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

[Tru04] Trusted Computing Group. TPM main specification version 1.2, 2004.

[Tru14] Trusted Computing Group. Trusted platform module library specification, family "2.0", 2014.

A Functionalities and Simulators

Setup

1. `Issuer Setup.` On input (SETUP, sid) from issuer $\mathcal{I}$
   - Output $(\mathsf{FORWARD}, (\mathsf{SETUP}, sid), \mathcal{V})$ to $\mathcal{S}$.

Join

1. `Join Request.` 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 $\mathcal{S}$.
2. `Join Proceed.` On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Output $(\mathsf{FORWARD}, (\mathsf{JOINPROCEED}, sid, jsid, attrs), \mathcal{I})$ to $\mathcal{S}$.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL}), \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{M}_i)$ to $\mathcal{S}$.

Verify

1. `Verify.` On input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL})$ from some party $\mathcal{V}$.
   - Output $(\mathsf{FORWARD}, (\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL}), \mathcal{V})$ to $\mathcal{S}$.

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$..
   - Output $(\mathsf{FORWARD}, (\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn), \mathcal{V})$ to $\mathcal{S}$.

Output

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

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

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

KeyGen

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{SETUP}, sid), \mathcal{I})$ from $\mathcal{F}$.
  • Give "$\mathcal{I}$" input (SETUP, sid).

Join

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

Sign

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

Verify

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{VERIFY}, sid, m, bsn, \sigma, \mathtt{RL}), \mathcal{V})$ from $\mathcal{F}$.
  • Give "$\mathcal{V}$" input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, \mathtt{RL})$.

Link

– Upon receiving $(\mathsf{FORWARD}, (\mathsf{LINK}, sid, lsid, \sigma, m, \sigma', m', bsn), \mathcal{V})$ from $\mathcal{F}$.
  • Give "$\mathcal{V}$" input $(\mathsf{LINK}, sid, lsid, \sigma, m, \sigma', m', bsn)$.

Fig. 4. 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 $(\mathsf{ALG}, sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from $\mathcal{S}$
   - Check that ver, link and identify are deterministic.
   - Store $(sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

1. `Join Request.` On input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ from host $\mathcal{H}_j$.
   - Output $(\mathsf{FORWARD}, (\mathsf{JOIN}, sid, jsid, \mathcal{M}_i), \mathcal{V})$ to $\mathcal{S}$.
2. `Join Proceed.` On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Output $(\mathsf{FORWARD}, (\mathsf{JOINPROCEED}, sid, jsid, attrs), \mathcal{V})$ to $\mathcal{S}$.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL}), \mathcal{V})$ to $\mathcal{S}$.
2. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{V})$ to $\mathcal{S}$.

Verify

1. `Verify.` On input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL})$ from some party $\mathcal{V}$.
   - Output $(\mathsf{FORWARD}, (\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL}), \mathcal{V})$ to $\mathcal{S}$.

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$..
   - Output $(\mathsf{FORWARD}, (\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn), \mathcal{V})$ to $\mathcal{S}$.

Output

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

Fig. 5. $\mathcal{F}$ for Game 4

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

KeyGen

– On input (SETUP, sid) from $\mathcal{F}$.
  - Parse sid as $\mathcal{I}, sid'$.
  - Give "$\mathcal{I}$" input (SETUP, sid).
  - When "$\mathcal{I}$" outputs $(\mathsf{SETUPDONE}, sid)$, $\mathcal{S}$ takes its private key x.
  - Define $\mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$ as follows: First, create a BBS+ signature on gsk and attribute values where the disclosed attributes are taken from p and the undisclosed attributes are set to dummy values. Next, the algorithm performs the real world signing algorithm (performing both the tasks from the host and the TPM).
  - Define $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL})$ as the real world verification algorithm, except that the private-key revocation check is ommitted.
  - Define $\mathsf{link}(\sigma, m, \sigma', m', bsn)$ as follows: Parse the signatures as $(A', nym, \pi, \{\pi_i\}) \leftarrow \sigma$, $(A'', nym', \pi', \{\pi'_i\}) \leftarrow \sigma'$, and output 1 iff $nym = nym'$.
  - Define $\mathsf{identify}(\sigma, m, bsn, gsk)$ as follows: parse σ as $(A', nym, \pi, \{\pi_i\})$ and check $gsk \in \mathbb{Z}_p$ and $nym = \mathsf{H}_1(bsn)^{gsk}$. If so, output 1, otherwise 0.
  - Define ukgen as follows: take $gsk \xleftarrow{\$} \mathbb{Z}_p$ and output gsk.
  - $\mathcal{S}$ sends $(\mathsf{ALG}, sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ to $\mathcal{F}$.

Corrupt $\mathcal{I}$

– $\mathcal{S}$ 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 is of the form $w, h_0, \ldots, h_L, \pi$ and π is valid, $\mathcal{S}$ extracts x from π.
  - $\mathcal{S}$ defines the algorithms $\mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen}$ as before, but now depending on the extracted key.
  - $\mathcal{S}$ sends (SETUP, sid') to $\mathcal{F}$ on behalf of $\mathcal{I}$.

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

– On input $(\mathsf{SETUPDONE}, sid)$ from $\mathcal{F}$
  - $\mathcal{S}$ continues simulating "$\mathcal{I}$".

Join

Unchanged.

Sign

Unchanged.

Verify

Unchanged.

Link

Unchanged.

Fig. 6. Simulator for Game 4

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 $(\mathsf{ALG}, sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from $\mathcal{S}$
   - Check that ver, link and $\mathsf{identify}$ are deterministic.
   - Store $(sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

1. `Join Request.` On input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ from host $\mathcal{H}_j$.
   - Output $(\mathsf{FORWARD}, (\mathsf{JOIN}, sid, jsid, \mathcal{M}_i), \mathcal{V})$ to $\mathcal{S}$.
2. `Join Proceed.` On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Output $(\mathsf{FORWARD}, (\mathsf{JOINPROCEED}, sid, jsid, attrs), \mathcal{V})$ to $\mathcal{S}$.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL}), \mathcal{V})$ to $\mathcal{S}$.
2. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{V})$ to $\mathcal{S}$.

Verify

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

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Output

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

Fig. 7. $\mathcal{F}$ for Game 5

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

KeyGen

Unchanged.

Join

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

Sign

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

Verify

Nothing to simulate.

Link

Nothing to simulate.

Fig. 8. Simulator for Game 5

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 $(\mathsf{ALG}, sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from $\mathcal{S}$
   - Check that ver, link and identify are deterministic.
   - Store $(sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

1. `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 $\mathcal{S}$.
2. `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 $\mathcal{I}$.
3. `Join Proceed.` On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ to $status \leftarrow complete$.
   - Output $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ to $\mathcal{S}$.
4. `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 $gsk \leftarrow \perp$.
   - Insert $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Members` and output $(\mathsf{JOINED}, sid, jsid)$ to $\mathcal{H}_j$.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL}), \mathcal{V})$ to $\mathcal{S}$.
2. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Output $(\mathsf{FORWARD}, (\mathsf{SIGNPROCEED}, sid, ssid), \mathcal{V})$ to $\mathcal{S}$.

Verify

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

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Output

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

Fig. 9. $\mathcal{F}$ for Game 6

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

KeyGen

Unchanged.

Join

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

- $\mathcal{S}$ receives $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - It simulates the real world protocol by giving "$\mathcal{H}_j$" input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ and waits for output $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ from "$\mathcal{I}$".
  - $\mathcal{S}$ 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 $\mathcal{H}$, $\mathcal{I}$, Corrupt $\mathcal{M}$:

- $\mathcal{S}$ receives $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - It simulates the real world protocol by giving "$\mathcal{H}_j$" input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ and waits for output $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ from "$\mathcal{I}$".
  - $\mathcal{S}$ extracts gsk from proof π_1 and stores it.
  - $\mathcal{S}$ 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 by giving "$\mathcal{H}_j$" input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ and waits for output $(\mathsf{JOINED}, sid, jsid)$ from "$\mathcal{H}_j$". Simulating "$\mathcal{H}_j$", $\mathcal{S}$ knows which attributes $attrs$ the corrupt issuer issued to "$\mathcal{H}_j$".
  - $\mathcal{S}$ sends $(\mathsf{JOINSTART}, sid, jsid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{JOINPROCEED}, sid, jsid)$ from $\mathcal{F}$.
  - $\mathcal{S}$ sends $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ to $\mathcal{F}$ on behalf of $\mathcal{I}$
- Upon receiving $(\mathsf{JOINCOMPLETE}, sid, jsid)$ from $\mathcal{F}$.
  - Send $(\mathsf{JOINCOMPLETE}, sid, jsid, \perp)$ to $\mathcal{F}$.

Fig. 10. First part of simulator for Game 6

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

- $\mathcal{S}$ notices this join as "$\mathcal{M}_i$" receives a $(sid, jsid, n)$ from $\mathcal{H}_j$
  - $\mathcal{S}$ makes a join query on $\mathcal{H}_j$'s behalf by sending $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ 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 $\mathcal{F}$, where gsk is taken from simulating "$\mathcal{M}_i$".
- Upon receiving $(\mathsf{JOINED}, sid, jsid)$ from $\mathcal{F}$ as $\mathcal{H}_j$ is corrupt.
  - $\mathcal{S}$ gives "$\mathcal{I}$" input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$.

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

- $\mathcal{S}$ notices this join as "$\mathcal{I}$" receives $(\mathsf{SENT}, sid', jsid, (Q, \pi_1), \mathcal{H}_j)$ from $\mathcal{F}_{\mathsf{auth}*}$.
  - Parse sid' as $(\mathcal{M}_i, \mathcal{I}, sid)$. $\mathcal{S}$ extracts gsk from π_1.
  - Note that $\mathcal{S}$ does not know the identity of the host that initiated this join, so it chooses some corrupt $\mathcal{H}_j$ and proceeds as if this is the host that initiated the join protocol. Even though this probably is not the correct host, it will only put a different host in `Members`, and the identities of hosts in this list are only used while creating signatures for platforms with an honest TPM or host, so for a fully corrupt platform it does not matter.
  - $\mathcal{S}$ makes a join query with $\mathcal{M}_i$ by sending $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ to $\mathcal{F}$ on behalf of $\mathcal{H}_j$.
- Upon receiving $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ 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 $\mathcal{H}_j$ is corrupt.
  - $\mathcal{S}$ 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 $(sid, jsid, n)$ from $\mathcal{H}_j$
- $\mathcal{S}$ simply simulates $\mathcal{M}_i$ honestly, there is no need to involve $\mathcal{F}$ as $\mathcal{M}_i$ does not receive inputs or send outputs in the join procedure.

Sign
Unchanged.
Verify
Nothing to simulate.
Link
Nothing to simulate.

Fig. 11. Second part of simulator for Game 6

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 $(\mathsf{ALG}, sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from $\mathcal{S}$
   - Check that ver, link and identify are deterministic.
   - Store $(sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, m, bsn, p, \mathtt{SRL}, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `Signature Generation.` On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i\rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i\rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk\rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$and store $\langle \mathcal{M}_i, bsn, gsk\rangle$ in $\mathtt{DomainKeys}$.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL}\rangle$ in $\mathtt{Signed}$.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

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

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 12. $\mathcal{F}$ for Game 7

KeyGen
Unchanged.
Join
Unchanged.
Sign
Honest $\mathcal{M}$, $\mathcal{H}$:

- Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, m, bsn, p, \mathtt{SRL}, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m', bsn', p, \mathtt{SRL})$, send $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - Continue the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, send $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.

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

- Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - Send $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ from $\mathcal{F}$ as $\mathcal{M}_i$ is corrupt.
  - Starts the simulation by giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, sends $(\mathsf{SIGNPROCEED}, sid, ssid)$ to $\mathcal{F}$ on behalf of $\mathcal{M}_i$.
- Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - Send $\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ to $\mathcal{F}$.

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

- $\mathcal{S}$ notices this sign as "$\mathcal{M}_i$" receives $sid, jsid, m, bsn, p, \mathtt{SRL}$ from $\mathcal{H}_j$.
  - Make a sign query on $\mathcal{H}_j$'s behalf by sending $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$.
- Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - Continue the simulation of "$\mathcal{M}_i$" until it outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$.
  - Sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - Send $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.
- As the host is corrupt, the platform may not have successfully joined, even when the TPM thinks it did. Therefore, the simulator either receives $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ from $\mathcal{F}$ when they joined, or $\perp$ as the membership check resulted in an error. On either one of those inputs:
  - Continue the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.

Verify
Nothing to simulate.
Link
Nothing to simulate.

Fig. 13. Simulator for Game 7

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 $(\mathsf{ALG}, sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from $\mathcal{S}$
   - Check that ver, link and identify are deterministic.
   - Store $(sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `Signature Generation.` On input $(\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ from $\mathcal{S}$.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i\rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i\rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk\rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$and store $\langle \mathcal{M}_i, bsn, gsk\rangle$ in $\mathtt{DomainKeys}$.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL}\rangle$ in $\mathtt{Signed}$.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

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

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 14. $\mathcal{F}$ for Game 8

KeyGen
Unchanged.
Join
Unchanged.
Sign
Honest $\mathcal{M}$, $\mathcal{H}$:

- Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - It takes dummy $m', bsn', p, \mathtt{SRL}$ such that $l(m', bsn', p, \mathtt{SRL}) = l$, making sure that p holds for the platform's attributes (which $\mathcal{S}$ learnt in the join protocol) and that $\mathtt{SRL}$ does not contain a signature matching the platform's gsk (again from the join protocol).
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m', bsn', p, \mathtt{SRL})$, send $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - Continue the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, send $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.

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

- Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - Send $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ from $\mathcal{F}$ as $\mathcal{M}_i$ is corrupt.
  - Starts the simulation by giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$.
  - When "$\mathcal{H}_j$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, sends $(\mathsf{SIGNPROCEED}, sid, ssid)$ to $\mathcal{F}$ on behalf of $\mathcal{M}_i$.
- Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - Send $\mathsf{SIGNCOMPLETE}, sid, ssid, \sigma)$ to $\mathcal{F}$.

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

- $\mathcal{S}$ notices this sign as "$\mathcal{M}_i$" receives $sid, jsid, m, bsn, p, \mathtt{SRL}$ from $\mathcal{H}_j$.
  - Make a sign query on $\mathcal{H}_j$'s behalf by sending $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$.
- Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - Continue the simulation of "$\mathcal{M}_i$" until it outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$.
  - Sends $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - Send $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.
- As the host is corrupt, the platform may not have successfully joined, even when the TPM thinks it did. Therefore, the simulator either receives $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ from $\mathcal{F}$ when they joined, or $\perp$ as the membership check resulted in an error. On either one of those inputs:
  - Continue the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.

Verify
Nothing to simulate.
Link
Nothing to simulate.

Fig. 15. Simulator for Game 8

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 $(\mathsf{ALG}, sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from $\mathcal{S}$
   - Check that ver, link and identify are deterministic.
   - Store $(sid, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ and output $(\mathsf{SETUPDONE}, sid)$ to $\mathcal{I}$.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i\rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i\rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk\rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$and store $\langle \mathcal{M}_i, bsn, gsk\rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL}\rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

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

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 16. $\mathcal{F}$ for Game 9

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 17. Simulator Game 9

Setup

Unchanged.

Join

1. `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 $\mathcal{S}$.
2. `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 $\mathcal{I}$.
3. `Join Proceed.` On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ to $status \leftarrow complete$.
   - 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.
4. `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 $gsk \leftarrow \perp$.
   - Insert $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Members` and output $(\mathsf{JOINED}, sid, jsid)$ to $\mathcal{H}_j$.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i\rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i\rangle \in$ `DomainKeys` where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk\rangle \in$ `DomainKeys`. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$and store $\langle \mathcal{M}_i, bsn, gsk\rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL}\rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

Unchanged.

Link

Unchanged.

Fig. 18. $\mathcal{F}$ for Game 10

KeyGen
Unchanged.
Join
Honest $\mathcal{M}$, $\mathcal{H}$, $\mathcal{I}$:

- $\mathcal{S}$ receives $(\mathsf{JOINSTART}, sid, jsid, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - It simulates the real world protocol by giving "$\mathcal{H}_j$" input $(\mathsf{JOIN}, sid, jsid, \mathcal{M}_i)$ and waits for output $(\mathsf{JOINPROCEED}, sid, jsid, \mathcal{M}_i)$ from "$\mathcal{I}$".
  - $\mathcal{S}$ sends $(\mathsf{JOINSTART}, sid, jsid)$ to $\mathcal{F}$.
- On input $(\mathsf{JOINCOMPLETE}, sid, jsid, attrs)$ from $\mathcal{F}$.
  - Note that $\mathcal{S}$ does not know the attributes, as it receives $attrs = \perp$. Pick a random $attrs'$ and continue the simulation.
  - $\mathcal{S}$ 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 $\mathcal{F}$.

Other cases:
Unchanged.
Sign
Honest $\mathcal{M}$, $\mathcal{H}$:

- Upon receiving $(\mathsf{SIGNSTART}, sid, ssid, l, \mathcal{M}_i, \mathcal{H}_j)$ from $\mathcal{F}$.
  - It takes dummy $m', bsn', p, \mathtt{SRL}$ such that $l(m', bsn', p, \mathtt{SRL}) = l$, making sure that p holds for the dummy attributes that $\mathcal{S}$ chose in the join protocol) and that $\mathtt{SRL}$ does not contain a signature matching the platform's gsk (again from the join protocol).
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{H}_j$" input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNPROCEED}, sid, ssid, m', bsn', p, \mathtt{SRL})$, send $(\mathsf{SIGNSTART}, sid, ssid)$ to $\mathcal{F}$.
- Upon receiving $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ from $\mathcal{F}$.
  - Continue the simulation by giving "$\mathcal{M}_i$" input $(\mathsf{SIGNPROCEED}, sid, ssid)$.
  - When "$\mathcal{M}_i$" outputs $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$, send $(\mathsf{SIGNCOMPLETE}, sid, ssid, \perp)$ to $\mathcal{F}$.

Other cases:
Unchanged.
Verify
Nothing to simulate.
Link
Nothing to simulate.

Fig. 19. Simulator for GAME 10

Setup

Unchanged.

Join

1. `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 $\mathcal{S}$.
2. `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 $\mathcal{I}$.
3. `Join Proceed.` On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ to $status \leftarrow complete$.
   - 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.
4. `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 by checking
     - $\mathsf{CheckGskHonest}(gsk) = 1$ if $\mathcal{H}_j$ is corrupt and $\mathcal{M}_i$ is honest, or
     - $\mathsf{CheckGskCorrupt}(gsk) = 1$ if $\mathcal{M}_i$ is corrupt.
   - Insert $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Members` and output $(\mathsf{JOINED}, sid, jsid)$ to $\mathcal{H}_j$.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i\rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i\rangle \in$ `DomainKeys` where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk\rangle \in$ `DomainKeys`. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk\rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL}\rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

Unchanged.

Link

Unchanged.

Fig. 20. $\mathcal{F}$ for Game 11

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 21. Simulator Game 11

Setup

Unchanged.

Join

1. `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 $\mathcal{S}$.
2. `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 $\mathcal{I}$.
3. `Join Proceed.` On input $(\mathsf{JOINPROCEED}, sid, jsid, attrs)$ from $\mathcal{I}$, with $attrs \in \mathbb{A}_1 \times \ldots \times \mathbb{A}_L$
   - Update the session record $\langle jsid, \mathcal{M}_i, \mathcal{H}_j, attrs, status\rangle$ to $status \leftarrow complete$.
   - 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.
4. `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 by checking
     - $\mathsf{CheckGskHonest}(gsk) = 1$ if $\mathcal{H}_j$ is corrupt and $\mathcal{M}_i$ is honest, or
     - $\mathsf{CheckGskCorrupt}(gsk) = 1$ if $\mathcal{M}_i$ is corrupt.
   - Insert $\langle \mathcal{M}_i, \mathcal{H}_j, gsk, attrs\rangle$ into `Members` and output $(\mathsf{JOINED}, sid, jsid)$ to $\mathcal{H}_j$.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs\rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status\rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i\rangle \in$ `Members` and $\langle \mathcal{M}_i, *, gsk_i\rangle \in$ `DomainKeys` where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk\rangle \in$ `DomainKeys`. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk\rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL}\rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

Unchanged.

Link

Unchanged.

Fig. 22. $\mathcal{F}$ for Game 12

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 23. Simulator Game 12

Setup

Unchanged.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that there are no two distinct gsk values matching σ'.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk \rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk \rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

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

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 24. $\mathcal{F}$ for Game 13

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 25. Simulator Game 13

Setup

Unchanged.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that there are no two distinct gsk values matching σ'.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk \rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk \rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

1. `Verify.` On input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL})$ from some party $\mathcal{V}$.
   - Retrieve all pairs $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma, m, bsn, gsk_i) = 1$. 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 $(gsk_i, \mathcal{M}_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in \mathtt{Members}$ exists with $p(attrs) = 1$.
     - There is a $gsk' \in \mathtt{RL}$ where $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL})$.
   - Add $\langle \sigma, m, bsn, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

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

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 27. Simulator Game 14

Setup

Unchanged.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that there are no two distinct gsk values matching σ'.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk \rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk \rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

1. `Verify.` On input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL})$ from some party $\mathcal{V}$.
   - Retrieve all pairs $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma, m, bsn, gsk_i) = 1$. 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 $(gsk_i, \mathcal{M}_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in \mathtt{Members}$ exists with $p(attrs) = 1$.
     - There is an honest $\mathcal{M}_i$ but no entry $\langle *, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle \in \mathtt{Signed}$ exists.
     - There is a $gsk' \in \mathtt{RL}$ where $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL})$.
   - Add $\langle \sigma, m, bsn, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 28. $\mathcal{F}$ for Game 15

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 29. Simulator Game 15

Setup

Unchanged.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that there are no two distinct gsk values matching σ'.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk \rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk \rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

1. `Verify.` On input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL})$ from some party $\mathcal{V}$.
   - Retrieve all pairs $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma, m, bsn, gsk_i) = 1$. 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 $(gsk_i, \mathcal{M}_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in \mathtt{Members}$ exists with $p(attrs) = 1$.
     - There is an honest $\mathcal{M}_i$ but no entry $\langle *, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle \in \mathtt{Signed}$ exists.
     - There is a $gsk' \in \mathtt{RL}$ where $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$ and no pair $(gsk_i, \mathcal{M}_i)$ for an honest $\mathcal{M}_i$ was found.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL})$.
   - Add $\langle \sigma, m, bsn, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 30. $\mathcal{F}$ for Game 16

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 31. Simulator Game 16

Setup

Unchanged.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that there are no two distinct gsk values matching σ'.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk \rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk \rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

1. `Verify.` On input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL})$ from some party $\mathcal{V}$.
   - Retrieve all pairs $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma, m, bsn, gsk_i) = 1$. 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 $(gsk_i, \mathcal{M}_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in \mathtt{Members}$ exists with $p(attrs) = 1$.
     - There is an honest $\mathcal{M}_i$ but no entry $\langle *, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle \in \mathtt{Signed}$ exists.
     - There is a $gsk' \in \mathtt{RL}$ where $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$ and no pair $(gsk_i, \mathcal{M}_i)$ for an honest $\mathcal{M}_i$ was found.
     - For some matching gsk_i and $(\sigma', m', bsn') \in \mathtt{SRL}$, $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL})$.
   - Add $\langle \sigma, m, bsn, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - Set $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 32. $\mathcal{F}$ for Game 17

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 33. Simulator Game 17

Setup

Unchanged.

Join

Unchanged.

Sign

1. `Sign Request.` On input $(\mathsf{SIGN}, sid, ssid, \mathcal{M}_i, m, bsn, p, \mathtt{SRL})$ from $\mathcal{H}_j$ with $p \in \mathbb{P}$
   - If $\mathcal{H}_j$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, *, attrs \rangle$ with $p(attrs) = 1$ exists in `Members`, abort.
   - Create a sign session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status \leftarrow request$.
   - Output $(\mathsf{SIGNSTART}, sid, ssid, l(m, bsn, p, \mathtt{SRL}), \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Sign Request Delivery.` On input $(\mathsf{SIGNSTART}, sid, ssid)$ from $\mathcal{S}$.
   - Update the session record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ to $status \leftarrow delivered$.
   - Output $(\mathsf{SIGNPROCEED}, sid, ssid, m, bsn, p, \mathtt{SRL})$ to $\mathcal{M}_i$.
3. `Sign Proceed.` On input $(\mathsf{SIGNPROCEED}, sid, ssid)$ from $\mathcal{M}_i$.
   - Look up record $\langle ssid, \mathcal{M}_i, \mathcal{H}_j, m, bsn, p, \mathtt{SRL}, status \rangle$ with $status = delivered$.
   - Output $(\mathsf{SIGNCOMPLETE}, sid, ssid)$ to $\mathcal{S}$.
4. `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 $p(attrs) = 1$ exists in `Members`.
   - For every $(\sigma', m', bsn') \in \mathtt{SRL}$, find all $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
     - Check that there are no two distinct gsk values matching σ'.
     - Check that no pair $(gsk_i, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established gsk:
     - Find gsk from $\langle \mathcal{M}_i, bsn, gsk \rangle \in \mathtt{DomainKeys}$. If no such gsk exists, set $gsk \leftarrow \mathsf{ukgen}()$, check $\mathsf{CheckGskHonest}(gsk) = 1$, and store $\langle \mathcal{M}_i, bsn, gsk \rangle$ in `DomainKeys`.
     - Compute signature $\sigma \leftarrow \mathsf{sig}(gsk, m, bsn, p, \mathtt{SRL})$, check $\mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL}) = 1$.
     - Check $\mathsf{identify}(\sigma, m, bsn, gsk) = 1$ and that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with key gsk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$.
   - If $\mathcal{M}_i$ is honest, store $\langle \sigma, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, sid, ssid, \sigma)$ to $\mathcal{H}_j$.

Verify

1. `Verify.` On input $(\mathsf{VERIFY}, sid, m, bsn, \sigma, p, \mathtt{RL}, \mathtt{SRL})$ from some party $\mathcal{V}$.
   - Retrieve all pairs $(gsk_i, \mathcal{M}_i)$ from $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{Members}$ and $\langle \mathcal{M}_i, *, gsk_i \rangle \in \mathtt{DomainKeys}$ where $\mathsf{identify}(\sigma, m, bsn, gsk_i) = 1$. 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 $(gsk_i, \mathcal{M}_i)$ was found for which an entry $\langle \mathcal{M}_i, *, *, attrs \rangle \in \mathtt{Members}$ exists with $p(attrs) = 1$.
     - There is an honest $\mathcal{M}_i$ but no entry $\langle *, m, bsn, \mathcal{M}_i, p, \mathtt{SRL} \rangle \in \mathtt{Signed}$ exists.
     - There is a $gsk' \in \mathtt{RL}$ where $\mathsf{identify}(\sigma, m, bsn, gsk') = 1$ and no pair $(gsk_i, \mathcal{M}_i)$ for an honest $\mathcal{M}_i$ was found.
     - For some matching gsk_i and $(\sigma', m', bsn') \in \mathtt{SRL}$, $\mathsf{identify}(\sigma', m', bsn', gsk_i) = 1$.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(\sigma, m, bsn, p, \mathtt{SRL})$.
   - Add $\langle \sigma, m, bsn, \mathtt{RL}, f \rangle$ to `VerResults` and output $(\mathsf{VERIFIED}, sid, f)$ to $\mathcal{V}$.

Link

1. `Link.` On input $(\mathsf{LINK}, sid, \sigma, m, p, \mathtt{SRL}, \sigma', m', p', \mathtt{SRL}', bsn)$ from a party $\mathcal{V}$.
   - Output $\perp$ to $\mathcal{V}$ if at least one signature $(\sigma, m, bsn, p, \mathtt{SRL})$ or $(\sigma', m', bsn, p', \mathtt{SRL}')$ is not valid (verified via the `verify` interface with $\mathtt{RL} = \emptyset$).
   - For each gsk_i in `Members` and `DomainKeys` compute $b_i \leftarrow \mathsf{identify}(\sigma, m, bsn, gsk_i)$ and $b_i' \leftarrow \mathsf{identify}(\sigma', m', bsn, gsk_i)$ and 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 $f \leftarrow \mathsf{link}(\sigma, m, \sigma', m', bsn)$.
   - Output (LINK, sid, f) to $\mathcal{V}$.

Fig. 34. $\mathcal{F}$ for Game 18

KeyGen
Unchanged.
Join
Unchanged.
Sign
Unchanged.
Verify
Unchanged.
Link
Unchanged.

Fig. 35. Simulator Game 18