Decentralized Traceable Attribute-Based Signatures

Ali El Kaafarani[1], Essam Ghadafi[2], and Dalia Khader[3]

[1] University of Bath, UK
[2] University of Bristol, UK
[3] Interdisciplinary Centre for Security, Reliability and Trust (SnT) , Luxembourg

Abstract. Attribute-based signatures allow a signer owning a set of attributes to anonymously sign a message w.r.t. some signing policy. A recipient of the signature is convinced that a signer with a set of attributes satisfying the signing policy has indeed produced the signature without learning the identity of the signer or which set of attributes was used in the signing.
Traceable attribute-based signatures add anonymity revocation mechanisms to attribute-based signatures whereby a special tracing authority equipped with a secret key is capable of revealing the identity of the signer. Such a feature is important in settings where accountability and abuse prevention are required.
In this work, we first provide a formal security model for traceable attribute-based signatures. Our focus is on the more practical case where attribute management is distributed among different authorities rather than relying on a single central authority. By specializing our model to the single attribute authority setting, we overcome some of the shortcomings of the existing model for the same setting.
Our second contribution is a generic construction for the primitive which achieves a strong notion of security. Namely, it achieves CCA anonymity and its security is w.r.t. adaptive adversaries. Moreover, our framework permits expressive signing polices. Finally, we provide some instantiations of the primitive whose security reduces to falsifiable intractability assumptions and without relying on idealized assumptions.

Keywords. Attribute-based signatures, security definitions, standard model.

1 Introduction

Attribute-based cryptography has emerged as an important research topic in recent years. It offers a versatile solution for designing role-based cryptosystems. In such systems, users are assigned attributes, and private operations (e.g. decryption/signing) are associated with security policies. Only users possessing attributes satisfying the policy in question can perform such operations. The first proposals of attribute-based cryptosystems were: an encryption scheme by Goyal et al. [22] (inspired by Sahai and Waters [41]) and a signature scheme by Maji et al. [35].

In traditional digital signature schemes, the recipient of a signature is convinced that a particular signer has indeed authenticated the message in question. In Attribute-Based Signatures (ABS) [35, 36], messages are signed w.r.t. a signing policy expressed as a predicate. Thus, the recipient is convinced that someone with a set of attributes satisfying the signing predicate has indeed authenticated the message without learning the identity of the signer or learning how the predicate was satisfied (i.e. which set of attributes was used in the signing).

There are many applications of attribute-based signatures such as attribute-based messaging, e.g. [8], trust negotiation, e.g. [15], and leaking secrets. Refer to [36] for more details and comparison with related notions such as mesh signatures [10] and anonymous credentials [11].

Besides correctness, the security of attribute-based signatures requires signer privacy and unforgeability. Informally, signer privacy (sometimes is also referred to as anonymity), requires that a signature reveals neither the identity of the signer nor which set of attributes was used to satisfy the associated predicate. On the other hand, unforgeability requires that a signer cannot forge a signature w.r.t. a signing predicate that her individual attributes do not satisfy, even if she colludes with other signers.

Traceable Attribute-Based Signatures (TABS) [13] extend standard attribute-based signatures by adding an anonymity revocation mechanism which allows a tracing authority to recover the identity of the signer if needed. This added feature is very important in scenarios where accountability and abuse prevention are required.

Related Work.Variants of attribute-based signatures exist in the literature each supporting policies that differ in their expressiveness. Those can be categorized into three main types of policies: non-monotonic policies, e.g. [38], monotonic policies, e.g. [36], and threshold-based policies, e.g. [33, 42, 32, 26, 18]. Schemes with more expressive policies are more interesting since they cover a larger scale of potential applications. Nevertheless, their current state–of–the–art instantiations are less efficient. The size of the signatures in existing instantiations of those supporting "monotonic" and "non-monotonic" policies, in the best case, are linearly dependent on the number of attributes in the policy [36, 38]. While the works of [26, 18] yield constant-size signatures, they only support threshold policies.

Early proposals of attribute-based signatures considered the case of multiple attribute authorities where each authority is responsible for a sub–universe of attributes [35, 38]. However, the multi–authority case still had the problem of relying on the existence of a central trusted authority. Moreover, in some cases, the security (unforgeability) of the whole system is compromised if the central authority is corrupted. Okamoto and Takashima [39] recently proposed the first decentralized construction.

Traceability in attribute-based signatures was first addressed by Khader [29] who proposed the notion of attribute-based group signatures. In this notion, only the anonymity of the identity of the signer is preserved, whereas the attributes used are not hidden. This is an undesirable property for many applications. Later, Khader et al. [30] proposed a traceable attribute-based signature scheme that relies on the verifier to decide the policy and thus requiring interaction in the signing protocol. Even though this can be useful in certain applications (see [30] for details), such interaction is prohibitive for many applications. A more recent construction by Escala et al. [13] adds the traceability feature (it was called revocation by the authors) to standard ABS schemes. The proposed scheme in [13] is in the inefficient composite-order groups setting and was originally proven in the Random Oracle Model (ROM) [4]. Although the authors described how the reliance on random oracles could be removed, this was done informally and without a concrete construction or a full security proof. In addition, their construction relies on a central attribute authority which could be a bottleneck when the number of members of the system increases.

Our Contribution. Our first contribution is a formal security model for traceable attribute-based signatures. Our focus is on the more interesting setting where there are multiple attribute authorities. We refer to this setting as Decentralized Traceable Attribute-Based Signatures (DTABS). By restricting the number of attribute authorities to one, we get a new model which addresses some of the shortcomings of the previous model for the same setting [13]. Our model can, in some sense, be viewed as an extension of the model of [39] where we allow signatures to be traced and allow for dynamic corruption of attribute authorities.

Our second contribution is a generic construction for DTABS. Our construction meets strong security requirements and permits expressive signing policies. Namely, it is CCA-anonymous under full-key exposure attacks, and its unforgeability is w.r.t. adaptively chosen messages and signing policies.

Finally, we present example instantiations of the generic construction and provide the first provably secure construction not relying on idealized assumptions. The security of all our instantiations rely on intractability assumptions which are falsifiable [37].

Paper Organization. In Section 2, we give some preliminary definitions. We formally define DTABS and provide their security model in Sections 3 and 4, respectively. We list the building blocks we use in Section 5. In Section 6, we present our generic construction and provide a proof of its security. In Section 7, we present constructions in the standard model.

Notation. A function $\nu(.) : \mathbb{N} \to \mathbb{R}^+$ is negligible in c if for every polynomial $p(.)$ and all sufficiently large values of c, it holds that $\nu(c) < \frac{1}{p(c)}$. Given a probability distribution S, we denote by $y \leftarrow S$ the operation of selecting an element according to S. If A is a probabilistic machine, we denote by $A(x_1, \ldots, x_n)$ the output distribution of A on inputs $(x_1, \ldots, x_n)$. By PPT we mean running in probabilistic polynomial time in the relevant security parameter.

2 Preliminaries

In this section we provide some preliminary definitions.

2.1 Bilinear Groups

A bilinear group is a tuple $\mathcal{P} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, G, \tilde{G}, e)$ where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of a prime order p and G and $\tilde{G}$ generate $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The function e is a non-degenerate bilinear map $\mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$. We will use multiplicative notation for all the groups although usually $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen to be additive groups. We let $\mathbb{G}_1^\times := \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$ and $\mathbb{G}_2^\times := \mathbb{G}_2 \setminus \{1_{\mathbb{G}_2}\}$. For clarity, elements from $\mathbb{G}_2$ will be accented with $\tilde{}$.

Following [19], we categorize prime-order bilinear groups into three main types:

- **Type-1**: This is the symmetric pairing setting in which $\mathbb{G}_1 = \mathbb{G}_2$.
- **Type-2**: $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is an efficiently computable isomorphism $\psi : \mathbb{G}_2 \longrightarrow \mathbb{G}_1$.
- **Type-3**: $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is no efficiently computable isomorphism between the groups in either direction.

We assume that there is an algorithm $\mathsf{BGrpSetup}$ that takes as input a security parameter λ and a type $\mathsf{tp} \in \{1, 2, 3\}$ and outputs a description of bilinear groups of Type-tp.

2.2 Complexity Assumptions

We will use the following assumptions from the literature:

DDH. For a group $\mathbb{G} := \langle G \rangle$ of a prime order p, given $(G, G^a, G^b, C) \in \mathbb{G}^4$ for $a, b \leftarrow \mathbb{Z}_p$, it is hard to decide whether or not $C = G^{ab}$.

SXDH. The Decisional Diffie-Hellman (DDH) assumption holds in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

DLIN [9]**.** For a group $\mathbb{G} := \langle G \rangle$ of a prime order p, given the tuple $(G^a, G^b, G^{ra}, G^{sb}, G^t)$ for unknown $a, b, r, s, t \in \mathbb{Z}_p$, it is hard to tell whether $t = r + s$ or t is random.

q**-SDH** [7]**.** For a group $\mathbb{G} := \langle G \rangle$ of a prime order p, given $(G, G^x, \ldots, G^{x^q})$ for $x \leftarrow \mathbb{Z}_p$, it is hard to output a pair $(c, G^{\frac{1}{x+c}}) \in \mathbb{Z}_p \times \mathbb{G}$ for an arbitrary $c \in \mathbb{Z}_p \backslash \{-x\}$.

WFCDH [16]**.** In symmetric bilinear groups, given $(G, G^a, G^b) \in \mathbb{G}^3$ for $a, b \leftarrow \mathbb{Z}_p$, it is hard to output a tuple $(G^r, G^{ra}, G^{rb}, G^{rab}) \in \mathbb{G}^{\times 4}$ for an arbitrary $r \in \mathbb{Z}_p$.

AWFCDH [16]**.** In asymmetric bilinear groups, given $(G, G^a, \tilde{G}) \in {\mathbb{G}_1}^2 \times \mathbb{G}_2$ for $a \leftarrow \mathbb{Z}_p$, it is hard to output a tuple $(G^b, G^{ab}, \tilde{G}^b, \tilde{G}^{ab}) \in {\mathbb{G}_1^\times}^2 \times {\mathbb{G}_2^\times}^2$ for an arbitrary $b \in \mathbb{Z}_p$.

q**-ADHSDH** [16]**.** In asymmetric bilinear groups [4], given $(G, F, K, G^x, \tilde{G}, \tilde{G}^x) \in \mathbb{G}_1^4 \times \mathbb{G}_2^2$ for $x \leftarrow \mathbb{Z}_p$, and $q-1$ tuples $(W_i := (K \cdot G^{u_i})^{\frac{1}{x+v_i}}, U_i := G^{u_i}, \tilde{U}_i := \tilde{G}^{u_i}, V_i := F^{v_i}, \tilde{V}_i := \tilde{G}^{v_i})_{i=1}^{q-1}$ for $u_i, v_i \leftarrow \mathbb{Z}_p$, it is hard to output a new tuple $(W^*, U^*, \tilde{U}^*, V^*, \tilde{V}^*)$ of this form.

[4] This can also be instantiated in symmetric groups. See [16].

2.3 Span Programs

For a field $\mathbb{F}$ and a variable set $A = \{a_1, \ldots, a_n\}$, a monotone span program [27] is define by a $\alpha \times \beta$ matrix $\mathbf{Z}$ (over $\mathbb{F}$) along with a labeling map ρ which associates each row in $\mathbf{Z}$ with an element $a_i \in A$.

The span program accepts a set γ iff $\mathbf{1} \in \text{Span}(\mathbf{Z}_\gamma)$, where $\mathbf{Z}_\gamma$ is the sub-matrix of $\mathbf{Z}$ containing only rows with labels $a_i \in \gamma$. In other words, the span program only accepts the set γ if there exists a vector $\boldsymbol{s}$ s.t. $\boldsymbol{s}\mathbf{Z}_\gamma = [1, 0, \ldots, 0]$.

3 Syntax of Decentralized Traceable Attribute-Based Signatures

The parties involved in a DTABS scheme are: κ attribute authorities each with a unique identity aid and a pair of secret/verification keys $(\mathsf{aask}_{\mathsf{aid}}, \mathsf{aavk}_{\mathsf{aid}})$; a tracing authority T which possesses a secret tracing key tk that can be used to trace the identity of the signer of a given signature; a set of signers each with a unique identity sid and a set of attributes $\mathcal{A} \subseteq \mathbb{A}$, where $\mathbb{A}$ is the universe of all possible attributes. An attribute can be uniquely identified by concatenating the identity of the managing attribute authority with the name of the attribute itself. A DTABS scheme is a tuple of polynomial-time algorithms

$$\mathsf{DTABS} := (\mathsf{Setup}, \mathsf{AuthSetup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Trace}, \mathsf{Judge}).$$

The syntax of the algorithms is defined below; where to aid notation all algorithms bar (Setup and $\mathsf{AuthSetup}$) are assumed to take as implicit input the public parameters pp output by algorithm Setup.

- $\mathsf{Setup}(1^\lambda)$ is run by some trusted third party. It takes as input a security parameter 1^λ and outputs public parameters pp and the tracing key tk. We assume that pp contains the attribute universe $\mathbb{A}$.
- $\mathsf{AuthSetup}(\mathsf{pp}, \mathsf{aid})$ used by attribute authority $\mathsf{Auth}_{\mathsf{aid}}$ to generate its key pair $(\mathsf{aask}_{\mathsf{aid}}, \mathsf{aavk}_{\mathsf{aid}})$. The attribute authority publishes its public verification key $\mathsf{aavk}_{\mathsf{aid}}$.
- $\mathsf{KeyGen}(\mathsf{aask}_{\mathsf{aid}}, \mathsf{sid}, a)$ takes as input an attribute authority's secret key $\mathsf{aask}_{\mathsf{aid}}$, a signer's identity sid and an attribute $a \in \mathbb{A}$ that signer sid possesses and generates a secret key $\mathsf{sk}_{\mathsf{sid},a}$ for attribute a for the signer. The key $\mathsf{sk}_{\mathsf{sid},a}$ is given to sid. The attribute authority may locally maintain a list of signers for which it ran the KeyGen algorithm.
- $\mathsf{Sign}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\mathcal{A}}, \{\mathsf{sk}_{\mathsf{sid},a}\}_{a\in\mathcal{A}}, m, \Psi)$ signer sid who possesses a set of attributes $\mathcal{A} \subseteq \mathbb{A}$ uses this algorithm to produce a signature on m w.r.t. the signing policy Ψ where $\Psi(\mathcal{A}) = 1$. The algorithm takes as input an ordered list of attribute authorities' verification keys $\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\mathcal{A}}$, an ordered list of attributes' secret keys $\{\mathsf{sk}_{\mathsf{sid},a}\}_{a\in\mathcal{A}}$, a message m and a signing predicate Ψ, and outputs a signature σ. Here $\mathsf{aid}(a)$ denotes the identity of the attribute authority managing attribute $a \in \mathbb{A}$.
- $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi)$ is a deterministic algorithm which takes as input an ordered list of attribute authorities' verification keys $\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}$, a message m, a signature σ and a signing predicate Ψ, and outputs 1 if σ is valid on m w.r.t. the signing predicate Ψ or 0 otherwise.
- $\mathsf{Trace}(\mathsf{tk}, m, \sigma, \Psi)$ is a deterministic algorithm which takes as input T's key tk, a message m, a signature σ and a signing predicate Ψ, and outputs the identity sid of the signer plus a proof π attesting to this claim. If the algorithm is unable to trace the signature to a signer, it returns the special symbol $\perp$. Note that if the tracing authority additionally gets a read-only access to the local registration tables maintained by the attribute authorities (whose identities can be inferred from the signing policy Ψ), then the tracing authority could additionally check whether or not sid has run the KeyGen algorithm.
- $\mathsf{Judge}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi, \mathsf{sid}, \pi)$ is a deterministic algorithm which takes as input an ordered list of attribute authorities' verification keys $\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}$, a message m, a signature σ, a signing predicate Ψ, a signer identity sid, and a tracing proof π, and outputs 1 if π is a valid proof that sid has produced σ or 0 otherwise.

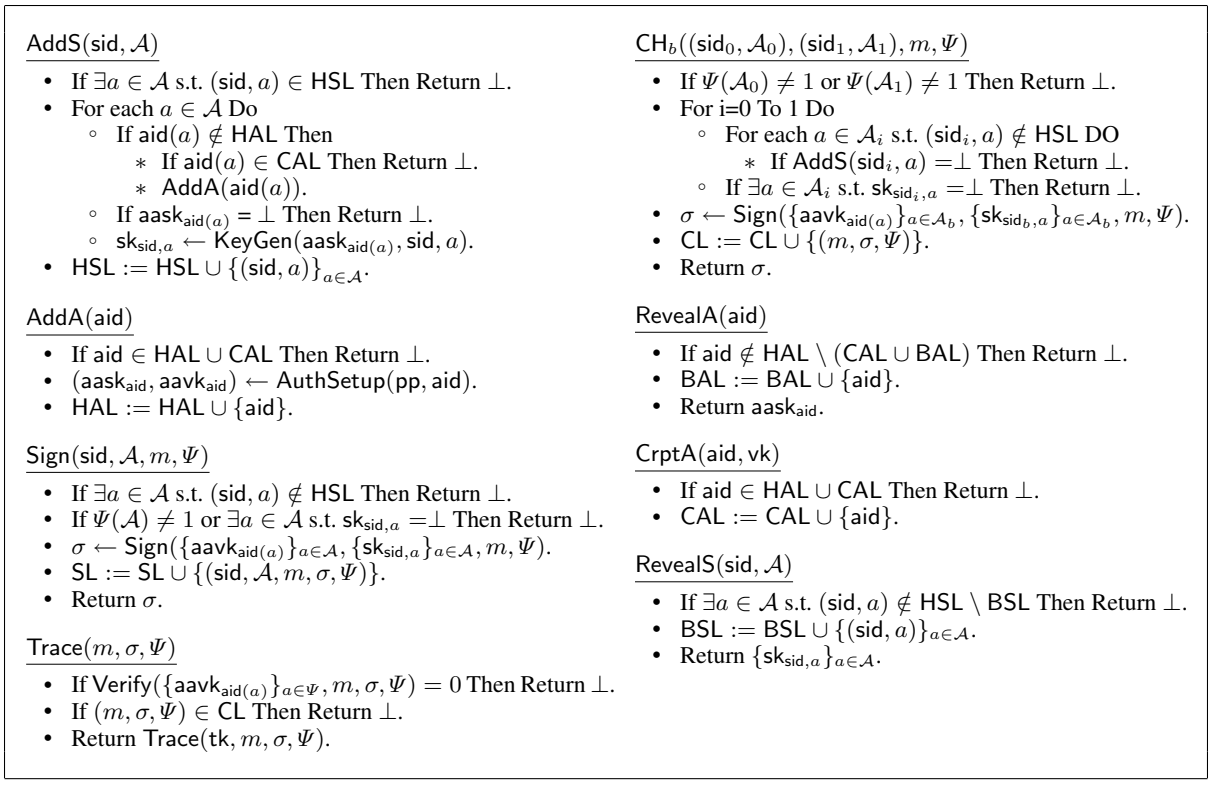

Fig. 1. Oracles used in the security games for DTABS

4 Security of Decentralized Traceble Attribute-Based Signatures

The security properties required from a DTABS scheme are: correctness, anonymity, full unforgeability, and traceability. In defining those requirements we use a set of experiments in which the adversary has access to a set of oracles. The following global lists are maintained: HSL is a list of honest signers' attributes and has entries of the form (sid, a); HAL is a list of honest attribute authorities; BSL is a list of bad signers' attributes whose secret keys have been revealed to the adversary with entries of the form (sid, a); BAL is a list of bad attribute authorities whose secret keys have been learned by the adversary; CAL is a list of corrupt attribute authorities whose keys have been chosen by the adversary; SL is a list of signatures obtained from the Sign oracle; CL is a list of challenge signatures obtained from the challenge oracle and is used only in the anonymity game.

The details of the following oracles are given in Fig. 1.

- AddS(sid, $\mathcal{A}$) is used to add honest attributes $\mathcal{A} \subseteq \mathbb{A}$ for signer sid. It can be called multiple times to add more attributes.
- AddA(aid) is used to add an honest attribute authority with identity aid.
- CrptA(aid, vk) is used to create a corrupt attribute authority whose secret key is chosen by the adversary.
- RevealS(sid, $\mathcal{A}$) is used to obtain the secret keys $\{\text{sk}_{\text{sid},a}\}_{a \in \mathcal{A}}$ corresponding to the subset of attributes $\mathcal{A} \subseteq \mathbb{A}$ owned by signer sid. It can be called multiple times.
- RevealA(aid) is used to obtain the secret key aask$_{\text{aid}}$ of the honest attribute authority aid.
- Sign(sid, $\mathcal{A}, m, \Psi$) is used to obtain a signature σ on message m by signer sid using $\{\text{sk}_{\text{sid},a}\}_{a \in \mathcal{A}}$ where $\Psi(\mathcal{A}) = 1$.

Experiment: $\mathsf{Exp}^{\text{Corr}}_{\mathsf{DTABS},\mathcal{F}}(\lambda)$

- $(\mathsf{pp}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathsf{HSL} := \emptyset$.
- $(\mathsf{sid}, \mathcal{A}, m, \Psi) \leftarrow \mathcal{F}(\mathsf{pp} : \mathsf{AddS}(\cdot,\cdot), \mathsf{AddA}(\cdot))$.
- If $\Psi(\mathcal{A}) \neq 1$ or $\mathcal{A} \not\subseteq \mathbb{A}$ Then Return 0.
- If $\exists a \in \mathcal{A}$ s.t. $(\mathsf{sid}, a) \notin \mathsf{HSL}$ or $\mathsf{sk}_{\mathsf{sid},a} = \perp$ or $\mathsf{aid}(a) \notin \mathsf{HAL}$ Then Return 0.
- $\sigma \leftarrow \mathsf{Sign}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\mathcal{A}}, \{\mathsf{sk}_{\mathsf{sid},a}\}_{a\in\mathcal{A}}, m, \Psi)$.
- If $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi) = 0$ Then Return 1.
- $(\mathsf{sid}', \pi) \leftarrow \mathsf{Trace}(\mathsf{tk}, m, \sigma, \Psi)$.
- If $\mathsf{sid}' \neq \mathsf{sid}$ Then Return 1.
- If $\mathsf{Judge}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi, \mathsf{sid}, \pi) = 0$ Then Return 1.
- Return 0.

Experiment: $\mathsf{Exp}^{\text{Anon-}b}_{\mathsf{DTABS},\mathcal{F}}(\lambda)$

- $(\mathsf{pp}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathsf{CAL}, \mathsf{HSL}, \mathsf{HAL}, \mathsf{BSL}, \mathsf{BAL}, \mathsf{CL} := \emptyset$.
- $b^* \leftarrow \mathcal{F}(\mathsf{pp} : \mathsf{AddS}(\cdot,\cdot), \mathsf{AddA}(\cdot), \mathsf{CrptA}(\cdot,\cdot), \mathsf{RevealS}(\cdot,\cdot), \mathsf{RevealA}(\cdot), \mathsf{CH}_b((\cdot,\cdot),(\cdot,\cdot),\cdot,\cdot), \mathsf{Trace}(\cdot,\cdot,\cdot))$.
- Return b^*.

Experiment: $\mathsf{Exp}^{\text{F-Unforge}}_{\mathsf{DTABS},\mathcal{F}}(\lambda)$

- $(\mathsf{pp}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathsf{CAL}, \mathsf{HSL}, \mathsf{HAL}, \mathsf{BSL}, \mathsf{BAL}, \mathsf{SL} := \emptyset$.
- $(m^*, \sigma^*, \Psi^*, \mathsf{sid}^*, \pi^*) \leftarrow \mathcal{F}(\mathsf{pp}, \mathsf{tk} : \mathsf{AddS}(\cdot,\cdot), \mathsf{AddA}(\cdot), \mathsf{CrptA}(\cdot,\cdot), \mathsf{RevealS}(\cdot,\cdot), \mathsf{RevealA}(\cdot), \mathsf{Sign}(\cdot,\cdot,\cdot,\cdot))$.
- If $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi^*}, m^*, \sigma^*, \Psi^*) = 0$ Then Return 0.
- If $\mathsf{Judge}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi^*}, m^*, \sigma^*, \Psi^*, \mathsf{sid}^*, \pi^*) = 0$ Then Return 0.
- Let $\mathcal{A}_{\mathsf{sid}^*}$ be the set of attributes owned by sid^* and managed by dishonest (i.e. $\in \mathsf{CAL} \cup \mathsf{BAL}$) attribute authorities.
- If $\exists \mathcal{A}$ s.t. $\{(\mathsf{sid}^*, a)\}_{a\in\mathcal{A}} \subseteq \mathsf{BSL}$ and $\Psi^*(\mathcal{A} \cup \mathcal{A}_{\mathsf{sid}^*}) = 1$ Then Return 0.
- If $\exists(\mathsf{sid}^*, \cdot, m^*, \sigma^*, \Psi^*) \in \mathsf{SL}$ Then Return 0.
- Return 1.

Experiment: $\mathsf{Exp}^{\text{Trace}}_{\mathsf{DTABS},\mathcal{F}}(\lambda)$

- $(\mathsf{pp}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathsf{CAL}, \mathsf{HSL}, \mathsf{HAL}, \mathsf{BSL}, \mathsf{BAL}, \mathsf{SL} := \emptyset$.
- $(m^*, \sigma^*, \Psi^*) \leftarrow \mathcal{F}(\mathsf{pp}, \mathsf{tk} : \mathsf{AddS}(\cdot,\cdot), \mathsf{AddA}(\cdot), \mathsf{RevealS}(\cdot,\cdot), \mathsf{Sign}(\cdot,\cdot,\cdot,\cdot))$.
- If $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi^*}, m^*, \sigma^*, \Psi^*) = 0$ Then Return 0.
- $(\mathsf{sid}^*, \pi) \leftarrow \mathsf{Trace}(\mathsf{tk}, m^*, \sigma^*, \Psi^*)$.
- If $\mathsf{sid}^* = \perp$ Then Return 1.
- If $\mathsf{Judge}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi^*}, m^*, \sigma^*, \Psi^*, \mathsf{sid}^*, \pi) = 0$ Then Return 1.
- If $(\mathsf{sid}, \cdot) \in \mathsf{HSL}$ Then Return 0.
- Return 1.

Fig. 2. Security experiments for decentralized traceable attribute-based signatures

- $\mathsf{CH}_b((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)$ is a left-right oracle for defining anonymity and is only called once. The adversary sends a couple of identities $(\mathsf{sid}_0, \mathcal{A}_0)$, $(\mathsf{sid}_1, \mathcal{A}_1)$, a message m and a signing policy Ψ. If $\Psi(\mathcal{A}_0) = \Psi(\mathcal{A}_1) = 1$, the oracle returns a signature on m using $\{\mathsf{sk}_{\mathsf{sid}_b,a}\}_{a\in\mathcal{A}_b}$ for $b \leftarrow \{0,1\}$.
- $\mathsf{Trace}(m, \sigma, \Psi)$ allows the adversary to ask for signatures to be traced.

The security requirements are defined by the games in Fig. 2.

Correctness. This demands that signatures produced by honest signers are accepted by the Verify algorithm and trace to the signer who produced them. Moreover, the Judge algorithm accepts the proof produced by the Trace algorithm. Formally, a DTABS scheme is correct if for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{F}$ have a negligible advantage $\mathsf{Adv}^{\text{Corr}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) := \Pr[\mathsf{Exp}^{\text{Corr}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1]$.

Anonymity. This requires that a signature reveals neither the identity of the signer nor the set of attributes used in the signing. This is a stronger notion than what is used in other settings, e.g. [29, 33], which only require that the identity of the signer remains anonymous. Thus, our definition ensures that a signature does not reveal more information other than what can be already inferred from the signing predicate itself.

In the game, the adversary chooses a message, a signing policy and two signers with two, possibly different, sets of attributes with the condition that both sets have to satisfy the signing policy. The adversary gets a signature by either signer and wins if it correctly guesses the signer.

Our model provides the adversary with strong capabilities, for instance, it can fully corrupt the attribute authorities and can ask for signers' secret keys to be revealed including the two signers it chooses for the challenge (and thus capturing full-key exposure attacks). Note that since the adversary can sign on behalf of any signer, it is redundant to provide the adversary with a sign oracle. The only restriction we impose on the adversary is that it may not query the Trace oracle on the challenge signature.

In CPA-anonymity [9], the adversary is not given access to the Trace oracle. On the contrary, in CCA-anonymity [5], the adversary can ask Trace queries at any stage of the game on any signature except the challenge signature. One can also consider a weaker non-adaptive variant of CCA-anonymity where the adversary can only ask Trace queries before it sees the challenge signature.

WLOG and in order to simplify the security proofs, we only allow the adversary a single call to the challenge oracle. We show in Appendix A that this is sufficient by showing a reduction from any adversary with a polynomial number of calls to the challenge oracle to one with a single call via a hybrid argument.

Also, our definition captures unlinkability because the adversary has access to all signers' secret keys and hence can produce signatures on behalf of any signer.

Formally, a DTABS scheme is anonymous if for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{F}$ have a negligible advantage $\mathsf{Adv}^{\text{Anon}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\text{Anon-0}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{Anon-1}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1]\right|$.

Full Unforgeability. This requirement captures unforgeability scenarios where the forgery opens to a particular signer. It ensures that even if signers collude and combine their attributes together, they cannot forge a signature that opens to a signer whose attributes do not satisfy the signing predicate. It also covers non-frameability and ensures that even if signers collude, they cannot frame a user who did not produce the signature.

Unlike the single attribute authority setting, here we allow the adversary to adaptively create corrupt attribute authorities and learn some of the honest authorities' secret keys as long as there is at least a single honest attribute authority managing one of the attributes required for satisfying the policy used in the forgery. In addition, we allow the adversary to fully corrupt the tracing authority.

Our definition is adaptive and allows the adversary to adaptively choose the predicate and the message on which it wants to produce the forgery rather than having to select the predicate at the start of the game. Also, note that we consider the stronger form of unforgeability, i.e. (strong unforgeability) where the adversary wins even if it manages to produce a new signature on a message/predicate pair that was queried to the sign oracle. We refer to this stronger definition as *Strong Full Unforgeability* and use the abbreviation SFU for short. The definition can, in a straightforward manner, be adapted to work for the weaker variant used in, e.g. [5, 36, 13], by requiring that the forgery is not on a message/predicate pair that was queried to the sign oracle. For the latter variant which we refer to as *Weak Full Unforgeability* (WFU), we just need to replace the check $\exists(\mathsf{sid}^*, \cdot, m^*, \Psi^*, \sigma^*) \in \mathsf{SL}$ by the check $\exists(\mathsf{sid}^*, \cdot, m^*, \Psi^*, \cdot) \in \mathsf{SL}$.

Formally, a DTABS scheme is fully unforgeable if for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{F}$ have a negligible advantage $\mathsf{Adv}^{\text{F-Unforge}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) := \Pr[\mathsf{Exp}^{\text{F-Unforge}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1]$.

Traceability. This requirement ensures that the adversary cannot produce a signature that traces to a signer who did not run the honest KeyGen algorithm. Thus, it covers unforgeability scenarios where the forgery is

untraceable. In the game, the adversary is allowed to corrupt the tracing authority and ask for the signing keys of any signer to be revealed. However, unlike in the full unforgeability game, we require that all the attribute authorities are honest as knowing a secret key of any attribute authority makes it easy to create signatures by dummy signers which are thus untraceable.

Formally, a DTABS scheme is traceable if for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{F}$ have a negligible advantage $\mathsf{Adv}^{\text{Trace}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) := \Pr[\mathsf{Exp}^{\text{Trace}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1]$.

4.1 Comparison with Escala et al. Model [13] for the Single Attribute Authority Setting

Specializing our model to the single attribute authority setting, we get a stronger model than the one in [13]. In particular, our model avoids some of the shortcomings inherent to [13] which we now explain. When defining non-frameability in [13], the sign oracle used by [13] does not consider the identity of the signer and hence it does not capture the following scenario: The adversary asks for two different signers sid_1 with attributes $\mathcal{A}_1$ and sid_2 with attributes $\mathcal{A}_2$ to be added. It then asks for a signature on the message m w.r.t. the signing policy Ψ by signer sid_1 (where $\Psi(\mathcal{A}_1) = 1$), and outputs as its forgery a signature σ^* on the same message m w.r.t. the same signing policy Ψ but the signature opens to sid_2 (assume here that $\Psi(\mathcal{A}_2) = 1$).

Therefore, we believe that in this context, where traceability is required, it is important that the identity of the signer is taken into account when answering signing queries. Otherwise, some of the unforgeability scenarios are not captured. This is, of course, different from standard attribute-based signatures where traceability is not required and thus there is no way for the adversary to learn who produced a particular signature.

In addition, our full unforgeability definition protects against a fully corrupt tracing authority which is stronger than the non-frameability definition in [13] which only considers a partially but not fully corrupt tracing authority.

5 Building Blocks

In this section we present the building blocks that we use in our constructions.

5.1 Tagged Signature Scheme

We define here a new variant of a signature scheme which we call a Tagged Signature (TS) scheme[5]. A tagged signature scheme for a message space $\mathcal{M}_{\mathsf{TS}}$ and a tag space $\mathcal{T}_{\mathsf{TS}}$ is a tuple of algorithms

$$\mathsf{TS} := ([\mathsf{Setup}], \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}) \ .$$

- $\mathsf{Setup}(1^\lambda)$ this optional algorithm takes as input a security parameter and outputs common public parameters param which is an implicit input to the rest of algorithms.
- $\mathsf{KeyGen}(\{\mathsf{param}|1^\lambda\})$ takes as input either public parameters (if the scheme requires a setup) or just the security parameter (if no setup is required) and outputs a pair of secret/verification keys $(\mathsf{sk}, \mathsf{vk})$.
- $\mathsf{Sign}(\mathsf{sk}, \tau, m)$ takes as input a secret key sk, a tag $\tau \in \mathcal{T}_{\mathsf{TS}}$ and a message $m \in \mathcal{M}_{\mathsf{TS}}$, and outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, \tau, m, \sigma)$ outputs 1 if σ is a signature on τ and m w.r.t. the verification key vk.

The security of a tagged signature scheme is similar to that of a traditional digital signature and consists of correctness and unforgeability:

[5] A variant of this primitive has been used by [1] in the context of one-time signatures which they call tagged one-time signatures.

- **Correctness:** Requires that for all $m \in \mathcal{M}_{\mathsf{TS}}$, $\tau \in \mathcal{T}_{\mathsf{TS}}$ and $(\mathsf{sk}, \mathsf{vk})$ output by KeyGen, we have $\mathsf{Verify}(\mathsf{vk}, \tau, m, \mathsf{Sign}(\mathsf{sk}, \tau, m)) = 1$.
- **(Existential) Unforgeability:** Unforgeability under adaptive chosen-message-tag attack requires that any PPT adversary $\mathcal{F}$ that is given a sign oracle $\mathsf{Sign}(\mathsf{sk}, \cdot, \cdot)$ has a negligible advantage in winning the following game:
  - A key pair $(\mathsf{sk}, \mathsf{vk})$ is generated and vk is sent to $\mathcal{F}$.
  - $\mathcal{F}$ makes a polynomial number of queries to $\mathsf{Sign}(\mathsf{sk}, \cdot, \cdot)$.
  - Eventually, $\mathcal{F}$ halts by outputting a tuple (σ^*, τ^*, m^*) and wins if σ^* is valid on (τ^*, m^*) and (τ^*, m^*) was never queried to Sign.

We note here that any signature scheme that can sign a pair of messages can be used as a tagged signature scheme. However, to allow for generality and explicitly distinguish the tag space from the message space (and hence care for the case where they might be distinct), we define this notion. Note that one can always use a collision-resistant hash function to map the tag into the message space.

Defining this as a new notion also serves to simplify the description of our constructions and security proofs. In particular, later in the constructions we need to hide the tag, whereas the message remains public and hence just signing the hash of the combination of the tag and the message using a standard digital signature would be problematic.

Instantiation. To construct a tagged signature, we use a variant of the automorphic scheme from [16] which was given in [17]. The original scheme given in [17] was given in the asymmetric setting. For simplicity, the variant we give here is in the symmetric setting. The tag space of the tagged signature scheme are Diffie–Hellman tuples $\mathcal{DH}$ where $\mathcal{DH} := \{(G^a, G'^a) \in \mathbb{G}^2 | a \in \mathbb{Z}_p\}$, whereas the message space is $\mathbb{Z}_p$. The scheme is unforgeable under the q-ADHSDH and WFCDH assumptions in the symmetric setting (and the q-ADHSDH and AWFCDH assumptions in the asymmetric setting) (cf. Section 2.2). Our tagged signature construction is as follows:

- $\mathsf{TS.Setup}(1^\lambda)$: Let $\mathcal{P} := (\mathbb{G}, \mathbb{G}_T, p, G, e)$ be the description of Type-1 bilinear groups. Choose $F, K, T, G', L \leftarrow \mathbb{G}$ and return $\mathsf{param} := (\mathcal{P}, F, K, T, L, G')$.
- $\mathsf{TS.KeyGen}(\mathsf{param})$: Choose $x \leftarrow \mathbb{Z}_p$ and set $(X, X') := (G^x, G'^x)$. Set $\mathsf{sk} := x$ and $\mathsf{vk} := (X, X')$.
- $\mathsf{TS.Sign}(\mathsf{sk}, (\tau, \tau'), m)$: Reject if $(\tau, \tau') \notin \mathcal{DH}$ (i.e. $e(\tau, G') \neq e(G, \tau')$). Otherwise, choose $u, v \leftarrow \mathbb{Z}_p$ and compute $\sigma := \left(U := G^u,\ U' := G'^u,\ V := F^v, V' := G'^v,\ W := (K \cdot T^u \cdot \tau \cdot L^m)^{\frac{1}{x+v}}\right)$.
- $\mathsf{TS.Verify}(\mathsf{vk}, (\tau, \tau'), m, \sigma)$: Check that $e(U, G') = e(G, U')$, $e(V, G') = e(F, V')$, and $e(W, X' \cdot V') = e(T, U')e(K \cdot \tau \cdot L^m, G')$, and output 1 or 0 accordingly.

5.2 The Full Boneh-Boyen (FBB) Signature Scheme

In [7], the authors gave a signature scheme that is secure under the q-SDH assumption (cf. Section 2.2). The signature scheme can be instantiated in both the symmetric and asymmetric bilinear group settings. Let $\mathcal{P} := (\mathbb{G}, \mathbb{G}_T, p, G, e)$ be the description of a bilinear group. The scheme is as follows; where to aid notation all algorithms bar KeyGen are assumed to take as implicit input $\mathcal{P}$:

- $\mathsf{KeyGen}(\mathcal{P})$: Choose $x, y \leftarrow \mathbb{Z}_p$ and set $(X, Y) := (G^x, G^y)$. Set $\mathsf{sk} := (x, y)$ and $\mathsf{vk} := (X, Y)$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: To sign $m \in \mathbb{Z}_p$, choose $r \leftarrow \mathbb{Z}_p$ such that $x + r \cdot y + m \neq 0$ and compute the signature $\sigma := G^{\frac{1}{x+r\cdot y+m}}$.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Output 1 if $e(\sigma, X \cdot Y^r \cdot G^m) = e(G, G)$ and 0 otherwise.

5.3 Strongly Unforgeable One-Time Signatures

A digital signature scheme is called *one-time signature* if in the unforgeability game, the adversary is restricted to a single signing query. *Strong Unforgeability* as opposed to weak unforgeability requires that the adversary cannot even forge a new signature on a message that she obtained a signature on from the signing oracle. In this paper, we will instantiate the one-time signature using the Full Boneh-Boyen signature scheme from Section 5.2.

5.4 Simulation-Sound Non-Interactive Zero-Knowledge Proofs

Let $\mathcal{R}$ be an efficiently computable relation. For pairs $(x, w) \in \mathcal{R}$, we call x the statement and w the witness. We define the language $\mathcal{L}$ as all the statements x in $\mathcal{R}$. A Simulation-Sound Non-Interactive Zero-Knowledge (SS-NIZK) proof system for $\mathcal{R}$ is defined by a tuple of algorithms

$$(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract}, \mathsf{SimSetup}, \mathsf{SimProve}) \ .$$

Setup takes as input a security parameter 1^λ and outputs a common reference string crs and an extraction key xk which allows for witness extraction. Prove takes as input (crs, x, w) and outputs a proof π that $(x, w) \in \mathcal{R}$. Verify takes as input (crs, x, π) and outputs 1 if the proof is valid, or 0 otherwise. $\mathsf{Extract}$ takes as input $(\mathsf{crs}, \mathsf{xk}, x, \pi)$ and outputs a witness. $\mathsf{SimSetup}$ takes as input a security parameter 1^λ and outputs a simulated reference string $\mathsf{crs}_{\mathsf{Sim}}$ and a trapdoor key tr that allows for proof simulation. $\mathsf{SimProve}$ takes as input $(\mathsf{crs}_{\mathsf{Sim}}, \mathsf{tr}, x)$ and outputs a simulated proof π_{Sim} without a witness.

We require: completeness, soundness, zero-knowledge and simulation-soundness which are all formally defined in Appendix B.

Groth-Sahai Proofs. Groth-Sahai (GS) proofs [25] are efficient non-interactive proofs in the Common Reference String (CRS) model. The GS system can be instantiated under different intractability assumptions with the SXDH-based instantiation being the most efficient [21].

The language for the system has the form

$$\mathcal{L} := \{\mathsf{statement} \mid \exists\, \mathsf{witness} : E_1(\mathsf{statement}, \mathsf{witness}), \ldots, E_n(\mathsf{statement}, \mathsf{witness}) \text{ hold } \},$$

where $E_i(\mathsf{statement}, \cdot)$ is one of the types of equation summarized in Fig. 3, where $X_1, \ldots, X_m, Y_1, \ldots, Y_n \in \mathbb{G}$, $x_1, \ldots, x_m, y_1, \ldots, y_n \in \mathbb{Z}_p$ are secret variables (hence underlined), whereas $A_i, T \in \mathbb{G}$, $a_i, b_i, k_{i,j}, t \in \mathbb{Z}_p$, $t_T \in \mathbb{G}_T$ are public constants. Note that in the asymmetric setting, there are two types of MSM equations depending on which group the elements belong to.

While one can fully extract parts of the witness which are group elements from Groth-Sahai proofs, we can only extract a one-way function of exponents as fully extracting an exponent element of the witness requires solving discrete logarithms. Also, for the proof to be zero-knowledge, all the equations have to have a trivial right-hand side. In particular, in the PPE case, we can only obtain zero-knowledge if either $t_T = 1$ or one knows $P1, \ldots, P_k$ and $Q_1, \ldots, Q_k$ such that $t_T = \prod_{i=1}^{k} e(P_i, Q_i)$.

The proof system has perfect completeness, perfect soundness, composable witness-indistinguishability/zero-knowledge. Note that the original proof system in [25] is not simulation-sound. We refer to [25] for the formal definitions and details of the instantiations.

NIZK Proofs in the Random Oracle Model. NIZK proofs in the random oracle model can be obtained by applying the Fiat–Shamir transformation [14] to interactive Σ-protocols to eliminate the interaction. In these proofs, a hash function which is modeled as a random oracle is used to compute the challenge.

- **Pairing Product Equation (PPE):** $\prod_{i=1}^{n} e(A_i, \underline{Y_i}) \cdot \prod_{i=1}^{m} \prod_{j=1}^{n} e(\underline{X_i}, \underline{Y_j})^{k_{i,j}} = t_T \cdot$
- **Multi-Scalar Multiplication Equation (MSME):** $\prod_{i=1}^{n} A_i^{\underline{y_i}} \prod_{i=1}^{m} \underline{X_i}^{b_i} \prod_{i=1}^{m} \prod_{j=1}^{n} \underline{X_i}^{k_{i,j}\underline{y_j}} = T \cdot$
- **Quadratic Equation (QE) in $\mathbb{Z}_p$:** $\sum_{i=1}^{n} a_i \underline{y_i} + \sum_{i=1}^{m} \underline{x_i} b_i + \sum_{i=1}^{m} \sum_{j=1}^{n} \underline{x_i}\underline{y_j} = t \cdot$

Fig. 3. Types of equations one can use Groth-Sahai proofs for

Experiment: $\mathsf{Exp}^{\text{IND-CCA-b}}_{\mathsf{PKE},\mathcal{F}}(\lambda)$:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
- $(m_0, m_1, \mathsf{st}_{\mathsf{find}}) \leftarrow \mathcal{F}_{\mathsf{find}}\,(\mathsf{pk} : \mathsf{Dec}(\mathsf{sk}, \cdot))$, where $|m_0| = |m_1|$.
- $C_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
- $b^* \leftarrow \mathcal{F}_{\mathsf{guess}}\left(\mathsf{st}_{\mathsf{find}}, C_b : \mathsf{Dec}^{C_b}(\mathsf{sk}, \cdot)\right)$, where Dec^{C_b} returns $\perp$ if queried on C_b.
- Return b^*.

Fig. 4. IND–CCA security game for PKE

5.5 Public-Key Encryption

A Public-Key Encryption (PKE) scheme is a tuple of polynomial-time algorithms

$$\mathsf{PKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}),$$

where the algorithms are defined as follows:

- $\mathsf{KeyGen}(1^\lambda)$: Takes a security parameter 1^λ and outputs a public key pk and a secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, m)$: Takes as input the public key pk and a message m, and outputs a ciphertext C.
- $\mathsf{Dec}(\mathsf{sk}, C)$: Takes as input the secret key sk and a ciphertext C, and outputs a message m.

Besides the usual correctness requirement, we require that the scheme is indistinguishable against adaptive chosen-ciphertext attacks (IND-CCA), which is defined by the game in Fig. 4.

We say that an encryption scheme is IND-CCA secure if for all $\lambda \in \mathbb{N}$, all polynomial-time adversaries $\mathcal{F}$ have a negligible advantage

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{PKE},\mathcal{F}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\text{IND-CCA-0}}_{\mathsf{PKE},\mathcal{F}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{IND-CCA-1}}_{\mathsf{PKE},\mathcal{F}}(\lambda) = 1]\right|$$

5.6 Tag-Based Encryption

A Tag-based Public-Key Encryption (TPKE) scheme [34] is similar to a public-key encryption scheme with the only difference being that both Enc and Dec algorithms take as an additional input a tag t.

More formally, a TPKE scheme for a message space $\mathcal{M}_{\mathsf{TPKE}}$ and a tag space $\mathcal{T}_{\mathsf{TPKE}}$ is a tuple of polynomial-time algorithms

$$\mathsf{TPKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}[, \mathsf{IsValid}]),$$

where the algorithms are defined as follows:

- $\mathsf{KeyGen}(1^\lambda)$: Takes a security parameter 1^λ and outputs a public key pk and a secret key sk.

Experiment: $\mathsf{Exp}^{\text{ST-WIND-CCA-b}}_{\mathsf{TPKE},\mathcal{F}}(\lambda)$:

- $(t^*, \mathsf{st}_{\mathsf{init}}) \leftarrow \mathcal{F}_{\mathsf{init}}\left(1^\lambda\right)$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
- $(m_0, m_1, \mathsf{st}_{\mathsf{find}}) \leftarrow \mathcal{F}_{\mathsf{find}}\left(\mathsf{st}_{\mathsf{init}}, \mathsf{pk} : \mathsf{Dec}^{t^*}(\mathsf{sk}, \cdot, \cdot)\right)$, where $|m_0| = |m_1|$.
- $C_{\text{tbe},b} \leftarrow \mathsf{Enc}(\mathsf{pk}, t^*, m_b)$.
- $b^* \leftarrow \mathcal{F}_{\mathsf{guess}}\left(\mathsf{st}_{\mathsf{find}}, C_{\text{tbe},b} : \mathsf{Dec}^{t^*}(\mathsf{sk}, \cdot, \cdot)\right)$, where Dec^{t^*} returns $\perp$ if queried on t^*.
- Return b^*.

Fig. 5. ST-WIND-CCA security game for TPKE

- $\mathsf{TPKE.KeyGen}(1^\lambda)$
  - $(\mathbb{G}, p) \leftarrow \mathsf{GrpSetup}(1^\lambda)$.
  - $K, L \leftarrow \mathbb{G}$; $f, h \leftarrow \mathbb{Z}_p$.
  - $F := G^f$, $H := G^h$.
  - $\mathsf{pk} := (G, F, H, K, L)$; $\mathsf{sk} := (f, h)$.
- $\mathsf{TPKE.Enc}(\mathsf{pk}, t, M)$
  - $r_1, r_2 \leftarrow \mathbb{Z}_p$.
  - $C_1 := F^{r_1}$; $C_2 := H^{r_2}$; $C_3 := G^{r_1+r_2} \cdot M$.
  - $C_4 := (G^t \cdot K)^{r_1}$; $C_5 := (G^t \cdot L)^{r_2}$.
  - $C_{\text{tbe}} := (C_1, C_2, C_3, C_4, C_5)$.
- $\mathsf{TPKE.Dec}(\mathsf{sk}, t, C_{\text{tbe}})$
  - If $\mathsf{TPKE.IsValid}(\mathsf{pk}, t, C_{\text{tbe}}) = 0$ Then return $\perp$
  - Parse C_{tbe} as $(C_1, C_2, C_3, C_4, C_5)$.
  - $M := C_3 \cdot C_1^{-1/f} C_2^{-1/h}$.
- $\mathsf{TPKE.IsValid}(\mathsf{pk}, t, C_{\text{tbe}})$
  - Parse C_{tbe} as $(C_1, C_2, C_3, C_4, C_5)$.
  - If $e(F, C_4) \neq e(C_1, G^t \cdot K)$ Or $e(H, C_5) \neq e(C_2, G^t \cdot L)$ Then Return 0.
  - Else Return 1.

Fig. 6. The tag-based encryption by Kiltz [31]

- $\mathsf{Enc}(\mathsf{pk}, t, m)$: Takes as input the public key pk, a tag $t \in \mathcal{T}_{\mathsf{TPKE}}$ and a message $m \in \mathcal{M}_{\mathsf{TPKE}}$, and outputs a ciphertext C_{tbe}.
- $\mathsf{Dec}(\mathsf{sk}, t, C_{\text{tbe}})$: Takes as input the secret key sk, a tag $t \in \mathcal{T}_{\mathsf{TPKE}}$ and a ciphertext C_{tbe}, and outputs a message m or the reject symbol $\perp$.
- $\mathsf{IsValid}(\mathsf{pk}, t, C_{\text{tbe}})$: This is an optional algorithm and is used to check whether a ciphertext is valid under the tag t w.r.t. the pubic key pk. It returns 1 or 0 accordingly.

Besides the usual correctness requirement, we require selective-tag weak indistinguishability against adaptive chosen-ciphertext attacks (ST-WIND-CCA), which is defined by the game in Fig. 5.

We say the scheme is ST-WIND-CCA secure if for all $\lambda \in \mathbb{N}$, all polynomial-time adversaries $\mathcal{F}$ have a negligible advantage

$$\mathsf{Adv}^{\text{ST-WIND-CCA}}_{\mathsf{TPKE},\mathcal{F}}(\lambda) := \left|\Pr[\mathsf{Exp}^{\text{ST-WIND-CCA-0}}_{\mathsf{TPKE},\mathcal{F}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{ST-WIND-CCA-1}}_{\mathsf{TPKE},\mathcal{F}}(\lambda) = 1]\right|$$

Kiltz [31] showed how to combine a ST-WIND-CCA secure TPKE scheme with a strongly unforgeable one-time signature scheme to obtain an IND-CCA2 secure PKE scheme. In the transformation the one-time signature verification key is used as a tag for the TPKE scheme and then the tag-based ciphertext C_{tbe} is signed with the one-time signature secret signing key.

Instantiation of ST-WIND-CCA Tag-Based Encryption. We use the selective-tag weakly secure CCA tag-based encryption scheme by Kiltz [31] which is secure under the DLIN assumption. The scheme is in Fig. 6.

In [28], it was shown that the tag-based scheme in Fig. 6 can be translated into both (Type-2 & Type-3) asymmetric bilinear group settings. The security of the scheme in the Type-3 setting relies on a variant of the DLIN assumption called the SDLIN assumption, in which the last element in the input tuple is provided

in both groups. However, the security of this variant requires that the message space is polynomial in the security parameter so that we can efficiently search when decrypting.

6 A Generic Construction for Decentralized Traceable Attribute-Based Signatures

In this section, we present our generic construction for decentralized traceable attribute-based signatures.

Overview of the construction. The tools we use in our generic construction are two NIZK proof systems NIZK_1 and NIZK_2, an IND-CCA2 secure public-key encryption scheme PKE, an existentially unforgeable tagged signature scheme TS, and an existentially unforgeable digital signature scheme DS with a message space $\mathcal{M}_{\mathsf{DS}}$. In addition, we need a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathcal{M}_{\mathsf{DS}}$.

We require that the NIZK_1 proof system, which will be used in the signing, is simulation-sound [40] and a proof of knowledge [12]. In fact, it is sufficient for it to be only one-time simulation-sound (see Appendix B). On the contrary, it suffices that NIZK_2 is a zero-knowledge proof system, i.e. we require neither simulation-soundness nor knowledge extractability from NIZK_2.

The Setup algorithm generates two separate common reference strings crs_1 and crs_2 for the NIZK systems NIZK_1 and NIZK_2, respectively. It also generates a key pair $(\mathsf{tvk}, \mathsf{tsk})$ for the digital signature scheme DS, and an encryption/decryption key pair $(\mathsf{epk}, \mathsf{esk})$ for the encryption scheme PKE. The public parameters of the system is set to $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{tvk}, \mathsf{epk}, \mathbb{A}, \mathcal{H})$, where $\mathbb{A}$ is the universe of attributes. The tracing authority's key is set to $\mathsf{tk} := \mathsf{esk}$.

When a new attribute authority joins the system, it creates a secret/verification key pair $(\mathsf{aask}_{\mathsf{aid}}, \mathsf{aavk}_{\mathsf{aid}})$ for the tagged signature scheme TS. To generate a signing key for attribute $a \in \mathbb{A}$ for signer sid, the managing attribute authority signs the signer identity sid (used as tag) along with the attribute a using her secret tagged signature signing key. The resulting signature is used as the secret key for that attribute by signer sid.

To sign a message m w.r.t. a signing policy Ψ, the signer first encrypts her identity sid using the encryption scheme PKE (and some randomness μ) to obtain a ciphertext C. She then computes, using the NIZK system NIZK_1, a proof π that she encrypted her identity correctly and that she either has a digital signature on the hash of the combination of the signing predicate, the message and the ciphertext containing her identity, i.e. $\mathcal{H}(\Psi, m, C)$, that verifies w.r.t. the verification key tvk or that she owns enough attributes to satisfy the original signing predicate Ψ in the form of tagged signatures on her identity and the attributes. For ease of composition and following [36], we refer to $\mathcal{H}(\Psi, m, C)$ as pseudo-attributes and denote them by $a_{\Psi,m,C}$. Note here that including the ciphertext as part of the encoding of the pseudo-attribute does not affect the signature size.

The extended predicate $\hat{\Psi}$ is proved via a span program (see Section 2.3) represented by the matrix $\mathbf{Z}$: the signer proves that she knows a secret vector $\boldsymbol{s} \in \mathbb{Z}_p^{|\hat{\Psi}|}$ s.t. $\boldsymbol{s}\mathbf{Z} = [1, 0, \ldots, 0]$. She also needs to show that she possesses a valid (tagged) signature on each attribute in the signing predicate for which the corresponding element in $\boldsymbol{s}$ is non-zero or a valid signature that verifies w.r.t tvk in the case of a pseudo-attribute. For attributes appearing in the policy that the signer does not own, she chooses random signatures.

Note that the hiding property of the NIZK_1 system ensures that the proof π does not reveal how the modified predicate $\hat{\Psi}$ was satisfied, i.e. whether the signer has a special signature on the pseudo-attribute or she owns enough attributes to satisfy the original predicate Ψ. The signature is then set to $\sigma := (\pi, C)$. To verify the signature, one just needs to verify the proof π.

The modified OR predicate $\hat{\Psi}$ serves to bind the signature to the message and the signing predicate. The secret signing key tsk for the digital signature scheme DS is only used as a trapdoor in the security proofs, and thus is not given to any authority. It allows its holder to simulate signatures and sign on behalf

- $\underline{\mathsf{Setup}(1^\lambda)}$
  - $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$ and $\mathsf{crs}_2 \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$.
  - $(\mathsf{tvk}, \mathsf{tsk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$ and $(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda; \rho)$.
  - Choose a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \mathcal{M}_{\mathsf{DS}}$.
  - Let $\mathsf{tk} := \mathsf{esk}$ and $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{tvk}, \mathsf{epk}, \mathbb{A}, \mathcal{H})$, where $\mathbb{A}$ is the attribute universe.
  - Return pp.
- $\underline{\mathsf{AuthSetup}(\mathsf{pp}, \mathsf{aid})}$
  - $(\mathsf{aavk}_{\mathsf{aid}}, \mathsf{aask}_{\mathsf{aid}}) \leftarrow \mathsf{TS.KeyGen}(1^\lambda)$.
  - Return $(\mathsf{aavk}_{\mathsf{aid}}, \mathsf{aask}_{\mathsf{aid}})$.
- $\underline{\mathsf{KeyGen}(\mathsf{aask}_{\mathsf{aid}(a)}, \mathsf{sid}, a)}$
  - $\mathsf{sk}_{\mathsf{sid},a} \leftarrow \mathsf{TS.Sign}(\mathsf{aask}_{\mathsf{aid}(a)}, \mathsf{sid}, a)$.
  - Return $\mathsf{sk}_{\mathsf{sid},a}$.
- $\underline{\mathsf{Sign}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\mathcal{A}}, \{\mathsf{sk}_{\mathsf{sid},a}\}_{a\in\mathcal{A}}, m, \Psi)}$
  - Return $\perp$ if $\Psi(\mathcal{A}) = 0$.
  - $C \leftarrow \mathsf{PKE.Enc}(\mathsf{epk}, \mathsf{sid}; \mu)$.
  - Let $\hat{\Psi} := \Psi \vee a_{\Psi,m,C}$ and $\mathbf{Z} \in \mathbb{Z}_p^{|\hat{\Psi}|,\beta}$ be the span program for $\hat{\Psi}$.
  - Let $\boldsymbol{a} := \{a_i\}_{i=1}^{|\hat{\Psi}|}$ denote the attributes appearing in $\hat{\Psi}$.
  - $\pi \leftarrow \mathsf{NIZK}_1.\mathsf{Prove}(\mathsf{crs}_1, \{\mathsf{sid}, \mu, \boldsymbol{s}, \{\sigma_{a_i}\}_{i=1}^{|\hat{\Psi}|}\} : (C, \{\mathsf{aavk}_{\mathsf{aid}(a_i)}\}_{i=1}^{|\hat{\Psi}|-1} \cup \mathsf{tvk}, \boldsymbol{a}) \in \mathcal{L}_1)$.
  - Return $\sigma := (\pi, C)$.
- $\underline{\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi)}$
  - Return $\mathsf{NIZK}_1.\mathsf{Verify}(\mathsf{crs}_1, \pi)$.
- $\underline{\mathsf{Trace}(\mathsf{tk}, m, \sigma, \Psi)}$
  - Return $(\perp, \perp)$ if $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi) = 0$.
  - $\mathsf{sid} \leftarrow \mathsf{PKE.Dec}(\mathsf{tk}, C)$.
  - $\pi_{\mathsf{Trace}} \leftarrow \mathsf{NIZK}_2.\mathsf{Prove}(\mathsf{crs}_2, \{\mathsf{tk}, \rho\} : (C, \mathsf{epk}, \mathsf{sid}) \in \mathcal{L}_2)$.
  - Return $(\mathsf{sid}, \pi_{\mathsf{Trace}})$.
- $\underline{\mathsf{Judge}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi, \mathsf{sid}, \pi_{\mathsf{Trace}})}$
  - If $(\mathsf{sid}, \pi_{\mathsf{Trace}}) = (\perp, \perp)$ Then Return $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi) = 0$.
  - Return $\mathsf{NIZK}_2.\mathsf{Verify}(\mathsf{crs}_2, \pi_{\mathsf{Trace}})$.

Fig. 7. The generic construction for DTABS

of any signer without knowing their secret keys by simply encrypting their identity and producing a signature on the pseudo-attribute associated with the message and the signing predicate. Note that even in the unlikely case that any of the pseudo-attributes happened to collide with a real attribute, this is not a problem since signatures associated with pseudo-attributes must verify w.r.t. tvk which is different from all attribute authorities' keys.

To trace a signature, the tracing authority just decrypts the ciphertext C to recover the signer's identity. It then produces a proof π_{Trace} using the NIZK system NIZK_2 to prove that the decryption was done correctly. To verify the tracing correctness, the judge just needs to verify the validity of the NIZK proof π_{Trace}.

The details of the construction are in Fig. 7, whereas the languages associated with the NIZK proofs used in the construction are as follows, where for clarity we underline the elements of the witness:

$$\mathcal{L}_1 : \Big\{ \big((C, \mathbf{vk} := \{\mathsf{aavk}_{\mathsf{aid}(a_i)}\}_{i=1}^{|\hat{\Psi}|-1} \cup \mathsf{tvk}, \boldsymbol{a} := \{a_i\}_{i=1}^{|\hat{\Psi}|}), (\mathsf{sid}, \mu, \boldsymbol{s}, \boldsymbol{\sigma} := \{\sigma_{a_i}\}_{i=1}^{|\hat{\Psi}|})\big) : \\ \Big(\underline{\boldsymbol{s}}\mathbf{Z} = [1, 0, \ldots, 0] \bigwedge_{i=1}^{|\hat{\Psi}|-1} \text{ if } \underline{s_i} \neq 0 \Rightarrow \mathsf{TS.Verify}(\mathsf{vk}_i, \underline{\mathsf{sid}}, a_i, \underline{\sigma_{a_i}}) = 1 \\ \wedge \text{ if } \underline{s_{|\hat{\Psi}|}} \neq 0 \Rightarrow \mathsf{DS.Verify}(\mathsf{tvk}, a_{\Psi,m,C}, \underline{\sigma_{a_{|\hat{\Psi}|}}}) = 1\Big) \wedge \mathsf{PKE.Enc}(\mathsf{epk}, \underline{\mathsf{sid}}; \underline{\mu}) = C \Big\}.$$

The witness consists of a signer identity sid, the randomness μ used in encrypting sid, a vector $\boldsymbol{s} \in \mathbb{Z}_p^{|\hat{\Psi}|}$, and signatures $\{\sigma_{a_i}\}_{i=1}^{|\hat{\Psi}|}$ s.t. the span program $\mathbf{Z}$ verifies w.r.t. to $\boldsymbol{s}$ and for every non-zero element s_i for $i \in \{1, \ldots |\boldsymbol{s}| - 1\}$, the tagged signature σ_{a_i} on sid (as a tag) and the attribute a_i (as a message) verifies w.r.t. the corresponding attribute authority verification key, and if $s_{|\boldsymbol{s}|} \neq 0$, the signature $\sigma_{|\hat{\Psi}|}$, i.e. the one on the special pseudo-attribute verifies w.r.t. the verification key tvk.

$$\mathcal{L}_2 : \Big\{ \big((C, \mathsf{epk}, \mathsf{sid}), (\mathsf{tk}, \rho)\big) : \mathsf{PKE.KeyGen}(1^\lambda; \underline{\rho}) = (\mathsf{epk}, \underline{\mathsf{tk}}) \ \wedge \ \mathsf{PKE.Dec}(\underline{\mathsf{tk}}, C) = \mathsf{sid} \Big\}.$$

The witness consists of the tracing key, i.e. the decryption key for PKE, and the randomness ρ (if any) used in the key generation of PKE s.t. the encryption/decryption key pair is correct and the ciphertext C decrypts to sid.

Note that if we encrypted the whole witness of π (rather than just the signer identity) then we could drop the requirement for NIZK_1 to be a proof of knowledge. The reason why we cannot afford to do this is two-fold: first, since the decryption key is used as a tracing key and signers do not have their own personal key pairs, this would mean that a dishonest tracing authority will be able to forge on behalf of an honest signer once it has opened a signature by them. Second, since in both the full unforgeability and traceability experiments, the adversary has access to the tracing key, it would mean that we can no longer sign using pseudo-attributes since the adversary will be able to learn what witness we used in producing a signature.

Also, note that for the construction to satisfy the stronger variant of full unforgeability (i.e. SFU) rather than WFU, NIZK_1 must additionally be strongly non-malleable in the sense that it is infeasible for the adversary to even output a new proof for a statement that it received a proof for. In particular, as noted by [23] if the proof system is simulation-sound extractable [23] then it is non-malleable.

Theorem 1. *The construction in Fig. 7 is a secure decentralized traceable attribute-based signature if the building blocks are secure w.r.t. their security requirements.*

The full proof of this theorem can be found in Appendix C.

We note here that instantiations of all the tools we require for the generic construction exist in the literature in both the random oracle and the standard models. In particular, we note that in the random oracle model we can instantiate the proof systems required using the Fiat–Shamir heuristics [14]. Our focus is, however, on constructions which do not rely on idealized assumptions. Before we proceed we note here that the size of the signature in [13], which requires random oracles and is over the inefficient composite-order bilinear groups, is $\mathbb{G}^{|\hat{\Psi}|+\beta+7}$. Note that the size of the group order of composite-order groups is about 10 times that of their prime-order counterparts at the same security level.

In order to improve the efficiency in the standard model, we present a construction in the next section that slightly deviates from the generic framework.

7 Constructions in the Standard Model

In order to get more efficient constructions in the standard model, we slightly deviate from the generic framework by dropping the requirement that NIZK_1 is simulation-sound. In particular, in our instantiations we will use the Groth-Sahai proof system (which is the only efficient non-interactive proof system not relying on random oracles) to instantiate both NIZK_1 and NIZK_2 systems. Note that Groth-Sahai proofs are malleable and therefore not simulation-sound. Although there exist transformations which make Groth-Sahai proofs simulation-sound, e.g. [23], unfortunately, all those transformations degrade the efficiency of the proofs. Also, note that the fact that one cannot efficiently extract exponents from Groth-Sahai proofs is

- $\underline{\mathsf{Setup}(1^\lambda)}$
  - $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$ and $(\mathsf{crs}_2, \mathsf{xk}_2) \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$.
  - $(\mathsf{tvk}, \mathsf{tsk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$ and $(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{TPKE.KeyGen}(1^\lambda; \rho)$.
  - Choose collision-resistant hash functions $\mathcal{H} : \{0,1\}^* \to \mathcal{M}_{\mathsf{DS}}$ and $\hat{\mathcal{H}} : \{0,1\}^* \to \mathcal{T}_{\mathsf{TPKE}}$.
  - Let $\mathsf{tk} := \mathsf{esk}$ and $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{tvk}, \mathsf{epk}, \mathbb{A}, \mathcal{H}, \hat{\mathcal{H}})$, where $\mathbb{A}$ is the attribute universe.
  - Return pp.
- $\underline{\mathsf{AuthSetup}(\mathsf{pp}, \mathsf{aid})}$
  - $(\mathsf{aavk}_{\mathsf{aid}}, \mathsf{aask}_{\mathsf{aid}}) \leftarrow \mathsf{TS.KeyGen}(1^\lambda)$.
  - Return $(\mathsf{aavk}_{\mathsf{aid}}, \mathsf{aask}_{\mathsf{aid}})$.
- $\underline{\mathsf{KeyGen}(\mathsf{aask}_{\mathsf{aid}(a)}, \mathsf{sid}, a)}$
  - $\mathsf{sk}_{\mathsf{sid},a} \leftarrow \mathsf{TS.Sign}(\mathsf{aask}_{\mathsf{aid}(a)}, \mathsf{sid}, a)$.
  - Return $\mathsf{sk}_{\mathsf{sid},a}$.
- $\underline{\mathsf{Sign}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\mathcal{A}}, \{\mathsf{sk}_{\mathsf{sid},a}\}_{a\in\mathcal{A}}, m, \Psi)}$
  - Return $\perp$ if $\Psi(\mathcal{A}) = 0$.
  - $(\mathsf{otsvk}, \mathsf{otssk}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$.
  - $C_{\mathsf{tbe}} \leftarrow \mathsf{TPKE.Enc}(\mathsf{epk}, \hat{\mathcal{H}}(\mathsf{otsvk}), \mathsf{sid}; \mu)$.
  - Let $\hat{\Psi} := \Psi \vee a_{\Psi, m, C_{\mathsf{tbe}}, \hat{\mathcal{H}}(\mathsf{otsvk})}$ and $\mathbf{Z} \in \mathbb{Z}_p^{|\hat{\Psi}|, \beta}$ be the span program for $\hat{\Psi}$.
  - Let $\boldsymbol{a} := \{a_i\}_{i=1}^{|\hat{\Psi}|}$ denote the attributes appearing in $\hat{\Psi}$.
  - $\pi \leftarrow \mathsf{NIZK}_1.\mathsf{Prove}(\mathsf{crs}_1, \{\mathsf{sid}, \mu, \boldsymbol{s}, \{\sigma_{a_i}\}_{i=1}^{|\hat{\Psi}|}\} : (\hat{\mathcal{H}}(\mathsf{otsvk}), C_{\mathsf{tbe}}, \{\mathsf{aavk}_{\mathsf{aid}(a_i)}\}_{i=1}^{|\hat{\Psi}|-1} \cup \mathsf{tvk}, \boldsymbol{a}) \in \mathcal{L}_1')$.
  - $\sigma_{\mathsf{ots}} \leftarrow \mathsf{OTS.Sign}(\mathsf{otssk}, (\pi, C_{\mathsf{tbe}}, \mathsf{otsvk}))$.
  - Return $\sigma := (\sigma_{\mathsf{ots}}, \pi, C_{\mathsf{tbe}}, \mathsf{otsvk})$.
- $\underline{\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi)}$
  - Parse σ as $(\sigma_{\mathsf{ots}}, \pi, C_{\mathsf{tbe}}, \mathsf{otsvk})$.
  - Return 1 if all the following verify; otherwise, return 0:
    - $\mathsf{OTS.Verify}(\mathsf{otsvk}, (\pi, C_{\mathsf{tbe}}, \mathsf{otsvk}), \sigma_{\mathsf{ots}}) = 1$.
    - $\mathsf{NIZK}_1.\mathsf{Verify}(\mathsf{crs}_1, \pi) = 1$.
    - $\mathsf{TPKE.IsValid}(\mathsf{epk}, \hat{\mathcal{H}}(\mathsf{otsvk}), C_{\mathsf{tbe}}) = 1$.
- $\underline{\mathsf{Trace}(\mathsf{tk}, m, \sigma, \Psi)}$
  - Return $(\perp, \perp)$ if $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi) = 0$.
  - $\mathsf{sid} \leftarrow \mathsf{TPKE.Dec}(\mathsf{tk}, \hat{\mathcal{H}}(\mathsf{otsvk}), C_{\mathsf{tbe}})$.
  - $\pi_{\mathsf{Trace}} \leftarrow \mathsf{NIZK}_2.\mathsf{Prove}(\mathsf{crs}_2, \{\mathsf{tk}, \rho\} : (\hat{\mathcal{H}}(\mathsf{otsvk}), C_{\mathsf{tbe}}, \mathsf{epk}, \mathsf{sid}) \in \mathcal{L}_2')$.
  - Return $(\mathsf{sid}, \pi_{\mathsf{Trace}})$.
- $\underline{\mathsf{Judge}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi, \mathsf{sid}, \pi_{\mathsf{Trace}})}$
  - If $(\mathsf{sid}, \pi_{\mathsf{Trace}}) = (\perp, \perp)$ Then Return $\mathsf{Verify}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\Psi}, m, \sigma, \Psi) = 0$.
  - Return $\mathsf{NIZK}_2.\mathsf{Verify}(\mathsf{crs}_2, \pi_{\mathsf{Trace}})$.

Fig. 8. Details of the second construction

not a problem in our case as we never need to be able to efficiently extract the exponent components of the witness.

We will start by describing the idea of the construction generically and then present the exact instantiations later. To eliminate the need for NIZK_1 to be simulation-sound, we apply a trick similar to that used by Groth in [24] where we sign the final signature with a strongly unforgeable one-time signature scheme OTS. We require that OTS is strongly existentially unforgeable against adaptive chosen-message attack. We use the corresponding one-time verification key as a tag for a selective-tag weakly IND-CCA (i.e. ST-WIND-CCA secure) tag-based encryption scheme TPKE with which we encrypt the user's identity sid. The rest of the tools are the same as in the generic construction in Section 6.

To map the one-time signature verification key into the tag space of the tag-based encryption, we require another collision-resistant hash function, $\hat{\mathcal{H}} : \{0,1\}^* \to \mathcal{T}_{\mathsf{TPKE}}$. In order to further bind the signature to the one-time signature verification key (i.e. the tag used for the ciphertext), we sign the one-time signature

verification key as a part of the pseudo-attribute, i.e. the pseudo-attribute now is $(\Psi, m, C_{\text{tbe}}, \hat{\mathcal{H}}(\mathsf{otsvk}))$, which we will denote by $a_{\Psi,m,C_{\text{tbe}},\hat{\mathcal{H}}(\mathsf{otsvk})}$. The one-time signature serves to prevent the adversary from transforming a signature that it received into another valid signature as it now must be able to forge a one-time signature in order to succeed. Moreover, the one-time signature gives us the added bonus of realizing the stronger variant of full unforgeability.

The general idea of this construction is given in Fig. 8, whereas the languages associated with the NIZK proofs used in the construction are as follows, where again the elements of the witness are underlined:

- Language $\mathcal{L}'_1$ is defined as

$$\mathcal{L}'_1 : \Big\{ \big((\hat{\mathcal{H}}(\mathsf{otsvk}), C_{\text{tbe}}, \mathbf{vk} := \{\mathsf{aavk}_{\mathsf{aid}(a_i)}\}_{i=1}^{|\hat{\Psi}|-1} \cup \mathsf{tvk}, \boldsymbol{a} := \{a_i\}_{i=1}^{|\hat{\Psi}|}), (\mathsf{sid}, \mu, \boldsymbol{s}, \boldsymbol{\sigma} := \{\sigma_{a_i}\}_{i=1}^{|\hat{\Psi}|})\big) : \\ \Big(\underline{\boldsymbol{s}}\mathbf{Z} = [1, 0, \ldots, 0] \bigwedge_{i=1}^{|\hat{\Psi}|-1} \text{ if } \underline{s_i} \neq 0 \Rightarrow \mathsf{TS.Verify}(\mathsf{vk}_i, \underline{\mathsf{sid}}, a_i, \underline{\sigma_{a_i}}) = 1 \\ \wedge \text{ if } \underline{s_{|\hat{\Psi}|}} \neq 0 \Rightarrow \mathsf{DS.Verify}(\mathsf{tvk}, a_{\Psi,m,C_{\text{tbe}},\hat{\mathcal{H}}(\mathsf{otsvk})}, \underline{\sigma_{a_{|\hat{\Psi}|}}}) = 1\Big) \\ \wedge \ \mathsf{TPKE.Enc}(\mathsf{epk}, \hat{\mathcal{H}}(\mathsf{otsvk}), \underline{\mathsf{sid}}; \underline{\mu}) = C_{\text{tbe}} \Big\}.$$

- Language $\mathcal{L}'_2$ is defined as

$$\mathcal{L}'_2 : \Big\{ \big((\hat{\mathcal{H}}(\mathsf{otsvk}), C_{\text{tbe}}, \mathsf{epk}, \mathsf{sid}), (\mathsf{tk}, \rho)\big) : \mathsf{TPKE.KeyGen}(1^\lambda; \underline{\rho}) = (\mathsf{epk}, \underline{\mathsf{tk}}) \\ \wedge \ \mathsf{TPKE.Dec}(\underline{\mathsf{tk}}, \hat{\mathcal{H}}(\mathsf{otsvk}), C_{\text{tbe}}) = \mathsf{sid} \Big\}.$$

We provide a proof for the following Theorem in Appendix D.

Theorem 2. *The construction in Fig. 8 is a secure decentralized traceable attribute-based signature if the building blocks are secure w.r.t. their security requirements.*

7.1 An Instantiation in Symmetric Groups

We use the instantiation of the tagged signature scheme from Section 5.1 and instantiate the digital signature DS used for pseudo-attributes with the full Boneh-Boyen signature scheme (cf. Section 5.2) both in the symmetric setting. Thus, we assume a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$. Note that we need not hide the integer component r of the full Boneh-Boyen signature when proving π as such a signature can only be generated by the simulator running the security game and hence r does not reveal any information about the attributes involved or the identity of the signer. In other words, in both the real signature and the simulated signature cases, r is chosen uniformly at random.

We use the selective-tag weakly IND-CCA tag-based encryption scheme by Kiltz [31] as illustrated in Fig. 6 to instantiate TPKE and instantiate the one-time signature with the full Boneh-Boyen signature in the symmetric setting.

We now give the specific details of the proofs invloved. Let $\mathbf{Z} \in \mathbb{Z}_p^{|\hat{\Psi}|,\beta}$ be the span program for $\hat{\Psi} := \Psi \vee a_{\Psi,m,C_{\text{tbe}},\hat{\mathcal{H}}(\mathsf{otsvk})}$. To sign, we need the following proofs:

- To prove that $\boldsymbol{s}\mathbf{Z} = [1, 0, \ldots, 0]$, we need to prove the following linear equations:

$$\sum_{i=1}^{|\hat{\Psi}|} (\underline{s_i} Z_{i,1}) = 1 \qquad \sum_{i=1}^{|\hat{\Psi}|} (\underline{s_i} Z_{i,j}) = 0, \text{ for } j = 2, \ldots, \beta \tag{1}$$

To prove that if $\underline{s_i} \neq 0 \Rightarrow \mathsf{TS.Verify}(\mathsf{vk}_i, \underline{\mathsf{sid}}, a_i, \underline{\sigma_{a_i}}) = 1$, one needs to raise each pairing involved in the signature verification equations to s_i. This will ensure that if $s_i \neq 0$ then the only way for the equations to verify is by having a valid signature on sid and a_i. On the other hand, when the user does not own attribute a_i, i.e. does not have a valid signature σ_{a_i}, then $s_i = 0$ and the equations will verify since each pairing will evaluate to 1. Based on the observation that computing the components U, U', V, V' of the tagged signature does not require knowledge of the secret signing key and hence even when the user does not have a valid signature on a_i can still choose random components U, U', V, V' of the correct form to satisfy the first two verification equations of the tagged signature. Thus, it is sufficient to use s_i only in the last equation of the tagged signature verification equations. This reduces the number of additional GS commitments and equations required and hence improves the efficiency.
For each of the first $|\hat{\Psi}| - 1$ rows in $\mathbf{Z}$, we prove:

$$\underline{\bar{T}_i} = T^{\underline{s_i}} \qquad \underline{\bar{G}'_i} = G'^{\underline{s_i}} \qquad \underline{\bar{W}_i} = \underline{W_i}^{\underline{s_i}} \tag{2}$$

$$e(\underline{U_i}, G'_i) = e(G_i, \underline{U'_i}) \quad e(\underline{V_i}, G'_i) = e(F_i, \underline{V'_i}) \quad e(\underline{\bar{W}_i}, X' \cdot \underline{V'_i}) = e(\underline{\bar{T}_i}, \underline{U'_i}) e(K \cdot \underline{\mathsf{sid}} \cdot L^{a_i}, \underline{\bar{G}'_i}) \tag{3}$$

For the last row in $\mathbf{Z}$, i.e. the pseudo-attribute, the proofs required are:

$$\underline{\bar{\sigma}} = \underline{\sigma}^{\underline{s_{|\hat{\Psi}|}}} \qquad \underline{\bar{G}} = G^{\underline{s_{|\hat{\Psi}|}}} \qquad e(\underline{\bar{\sigma}}, X \cdot Y^r \cdot G^{\mathcal{H}(\Psi, m, C_{\mathsf{tbe}}, \hat{\mathcal{H}}(\mathsf{otsvk}))}) e(\underline{\bar{G}}, G) = 1 \tag{4}$$

- To prove that $\mathsf{TPKE.Enc}(\mathsf{epk}, \hat{\mathcal{H}}(\mathsf{otsvk}), \underline{\mathsf{sid}}; (\underline{r_1}, \underline{r_2})) = C_{\mathsf{tbe}}$, the signer needs to prove that she computed the ciphertext $(C_1, C_2, C_3, C_4, C_5) = \left(F^{r_1}, H^{r_2}, G^{r_1 + r_2} \cdot \mathsf{sid}, (G^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot K)^{r_1}, (G^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot L)^{r_2}\right)$ correctly. Since the validity of the ciphertext is publicly verifiable, and for the sake of efficiency, it is sufficient to provide proofs that C_1, C_2 and C_3 were computed correctly and the rest can be verified by checking that $e(F, C_4) = e(C_1, G^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot K)$ and $e(H, C_5) = e(C_2, G^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot L)$. Thus, proving this clause requires proving the 3 following multi-scalar multiplication equations, where the first two are linear, whereas the last equation is quadratic

$$C_1 = F^{\underline{r_1}} \qquad C_2 = H^{\underline{r_2}} \qquad C_3 = G^{\underline{r_1}} \cdot G^{\underline{r_2}} \cdot \underline{\mathsf{sid}} \tag{5}$$

  Note that we do not need to efficiently extract the exponents r_1 and r_2 from the proofs. Also, note that the equations are simultable and thus yield zero-knowledge proofs.
- Finally, the signer needs to prove that her identity is a Diffie–Hellman tuple satisfying $e(\underline{\mathsf{sid}}, G') = e(G, \underline{\mathsf{sid}'})$.

The total size of the Groth-Sahai proofs used is $\mathbb{Z}_p^{2 \cdot \beta} + \mathbb{G}^{42 \cdot |\hat{\Psi}| - 5}$. The proofs require $9 \cdot |\hat{\Psi}| - 1$ GS commitments each of size $\mathbb{G}^3$. The size of the tag-based ciphertext is $\mathbb{G}^5$, whereas the size of the one-time signature including the verification key is $\mathbb{G}^3 + \mathbb{Z}_p$. Thus, the total size of the signature is $\mathbb{Z}_p^{2 \cdot \beta + 1} + \mathbb{G}^{69 \cdot |\hat{\Psi}|}$. An important observation is that the verification of the signature could be made more efficient by using batch verification techniques for Groth-Sahai proofs [20, 6].

Tracing. Computing the proof π_{Trace} requires proving the following equations

$$G^{\underline{f}} = F \qquad G^{\underline{h}} = H \qquad C_3 \cdot C_1^{-1/\underline{f}} \cdot C_2^{-1/\underline{h}} = \mathsf{sid} \tag{6}$$

Those are 3 linear MSME proofs and immediately yield zero-knowledge. The total size of π_{Trace} is $\mathbb{G}^{12}$.

The proof for the following Theorem follows from that of Theorem 2.

Theorem 3. *The construction is secure if the assumptions DLIN, q-SDH, q-ADHSDH, and WFCDH hold.*

7.2 An Instantiation in Asymmetric Groups

To improve efficiency, here we translate the above instantiation into the asymmetric setting (i.e. Type-3 bilinear groups) where we use the more efficient SXDH-based instantiation of Groth-Sahai proofs. We use the asymmetric variants of all the building blocks used in the symmetric instantiation. Note that the security of the asymmetric instantiation of the tag-based encryption scheme from [28] which we use here is based on the SDLIN assumption [28] (a variant of the DLIN assumption in which the last element in the input tuple is provided in both groups) requires that the message space of the encryption scheme (i.e. the number of signers' identities to be encrypted) is polynomial in the security parameter so that we can efficiently search when decrypting. Thus, this instantiation only works when traceability is defined w.r.t. registered users in the system which is polynomial in the security parameter.

For clarity, in the following description we will also accent exponents with $\tilde{}$ when they are to be committed to in group $\mathbb{G}_2$. Let $\mathbf{Z} \in \mathbb{Z}_p^{|\hat{\Psi}|,\beta}$ be the span program for $\hat{\Psi} := \Psi \vee a_{\Psi,m,C_{\mathsf{tbe}},\hat{\mathcal{H}}(\mathsf{otsvk})}$. To sign, we need the following proofs:

- Commit to the vector $\boldsymbol{s}$ in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and prove that the values are equal which involves $|\hat{\Psi}|$ proofs for QEs of the form $s_i - \tilde{s_i} = 0$.
- To prove that $\boldsymbol{s}\mathbf{Z} = [1, 0, \ldots, 0]$, we need to prove the following linear equations:

$$\sum_{i=1}^{|\hat{\Psi}|}(\underline{s_i} Z_{i,1}) = 1 \qquad \sum_{i=1}^{|\hat{\Psi}|}(\underline{s_i} Z_{i,j}) = 0, \text{ for } j = 2, \ldots, \beta \tag{7}$$

  To prove that if $\underline{s_i} \neq 0 \Rightarrow \mathsf{TS.Verify}(\mathsf{vk}_i, \underline{\mathsf{sid}}, a_i, \underline{\sigma_{a_i}}) = 1$, for each of the first $|\hat{\Psi}| - 1$ rows in $\mathbf{Z}$, we prove:

$$\underline{\bar{T_i}} = T^{\underline{\tilde{s_i}}} \qquad \underline{\bar{\tilde{G}_i}} = \tilde{G}^{\underline{s_i}} \qquad \underline{\bar{W}_i} = \underline{W_i}^{\underline{\tilde{s_i}}} \tag{8}$$

$$e(\underline{U_i}, \tilde{G}_i) = e(G_i, \underline{\tilde{U}_i}) \qquad e(\underline{V_i}, \tilde{G}_i) = e(F_i, \underline{\tilde{V}_i}) \qquad e(\underline{\bar{W}_i}, \tilde{X} \cdot \underline{\tilde{V}_i}) = e(\underline{\bar{T}_i}, \underline{\tilde{U}_i}) e(K \cdot \underline{\mathsf{sid}} \cdot L^{a_i}, \underline{\bar{\tilde{G}}_i}) \tag{9}$$

  For the last row in $\mathbf{Z}$, i.e. the pseudo-attribute, the proofs required are:

$$\underline{\bar{\sigma}} = \underline{\sigma}^{\underline{\tilde{s}_{|\hat{\Psi}|}}} \qquad \underline{\bar{G}} = G^{\underline{\tilde{s}_{|\hat{\Psi}|}}} \qquad e(\underline{\bar{\sigma}}, \tilde{X} \cdot \tilde{Y}^r \cdot \tilde{G}^{\mathcal{H}(\Psi,m,C_{\mathsf{tbe}},\hat{\mathcal{H}}(\mathsf{otsvk}))}) e(\underline{\bar{G}}, \tilde{G}) = 1 \tag{10}$$

- To prove that $\mathsf{TPKE.Enc}(\mathsf{epk}, \hat{\mathcal{H}}(\mathsf{otsvk}), \underline{\mathsf{sid}}; (\underline{r_1}, \underline{r_2})) = C_{\mathsf{tbe}}$, the signer needs to prove that she computed the ciphertext $(C_1, C_2, C_3, \tilde{C_4}, \tilde{C_5}) = (F^{r_1}, H^{r_2}, G^{r_1+r_2} \cdot \mathsf{sid}, (\tilde{G}^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot \tilde{K})^{r_1}, (\tilde{G}^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot \tilde{L})^{r_2})$ correctly. Since the validity of the ciphertext is publicly verifiable, and for the sake of efficiency, it is sufficient to provide proofs that C_1, C_2 and C_3 were computed correctly and the rest can be verified by checking that $e(F, \tilde{C_4}) = e(C_1, \tilde{G}^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot \tilde{K})$ and $e(H, \tilde{C_5}) = e(C_2, \tilde{G}^{\hat{\mathcal{H}}(\mathsf{otsvk})} \cdot \tilde{L})$. Thus, proving this clause requires proving the 3 following multi-scalar multiplication equations, where the first two are linear, whereas the last equation is quadratic

$$C_1 = F^{\underline{\tilde{r_1}}} \qquad C_2 = H^{\underline{\tilde{r_2}}} \qquad C_3 = G^{\underline{\tilde{r_1}}} \cdot G^{\underline{\tilde{r_2}}} \cdot \underline{\mathsf{sid}} \tag{11}$$

  Note that we do not need to efficiently extract the exponents $\tilde{r_1}$ and $\tilde{r_2}$ from the proofs. Also, note that the equations are simultable and thus yield zero-knowledge proofs.

- Finally, the signer needs to prove that her identity is a Diffie–Hellman tuple satisfying $e(\underline{\mathsf{sid}}, \tilde{G}) = e(G, \underline{\tilde{\mathsf{sid}}})$.

The total size of the signature in this setting is $\mathbb{G}_1^{34\cdot|\hat{\Psi}|-6} + \mathbb{G}_2^{32\cdot|\hat{\Psi}|} + \mathbb{Z}_p^{\beta+1}$. Again, the verification of the signature could be made more efficient by using batch verification techniques for Groth-Sahai proofs [20, 6].

Tracing. Computing the proof π_{Trace} requires proving the following equations

$$G^{\underline{\tilde{f}}} = F \qquad\qquad G^{\underline{\tilde{h}}} = H \qquad\qquad C_3 \cdot C_1^{-1/\underline{\tilde{f}}} \cdot C_2^{-1/\underline{\tilde{h}}} = \mathsf{sid} \tag{12}$$

Those are 3 linear MSME proofs and immediately yield zero-knowledge. The total size of π_{Trace} is $\mathbb{G}_1^3 \times \mathbb{G}_2^4$.

The proof for the following Theorem follows from that of Theorem 2.

Theorem 4. *The construction is secure for a polynomial (in λ) signer identity space if the assumptions SDLIN in $\mathbb{G}_1$, q-SDH, q-ADHSDH, AWFCDH, and SXDH hold.*

We end by noting (similarly to [28]) that by translating the instantiation into the Type-2 setting, we can eliminate the requirement for the signer identity space (i.e. the message space of the TPKE scheme) to be polynomial. In this setting, we can use the instantiation of Groth-Sahai proofs based on DDH in $\mathbb{G}_1$ and DLIN in $\mathbb{G}_2$ as in [21].

7.3 Other Instantiations

By replacing the tagged signature scheme used in the previous instantiation with one based on any structure-preserving signature scheme [2] that is capable of signing two messages, we get more instantiations in the standard model.

An obvious candidate for this is the signature scheme by Abe et al. [3], which can sign multiple group elements and is at the same time compatible with Groth-Sahai proofs. The Abe et al. signature scheme yields signatures consisting of 7 group elements and requires 2 PPE for verification.

Acknowledgments. The second author was supported by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO and EPSRC via grant EP/H043454/1.

We thank David Bernhard, Liqun Chen, James Davenport, Nigel Smart, and Bogdan Warinschi. We also thank anonymous CT-RSA reviewers for valuable comments.

References

1. M. Abe, S. S. M. Chow, K. Haralambiev, and M. Ohkubo. Double-trapdoor anonymous tags for traceable signatures. In International Journal of Information Security, Springer LNCS 12, Issue 1, 19–31, 2013.
2. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *Crypto 2010*, Springer LNCS 6223, 209–236, 2010.
3. M. Abe, K. Haralambiev and M. Ohkubo. Signing on Elements in Bilinear Groups for Modular Protocol Design. *Cryptology ePrint Archive, Report 2010/133*. `http://eprint.iacr.org/2010/133`.
4. M. Bellare and P. Rogaway. Random oracles are practical: A Paradigm for Designing Efficient Protocols. In *ACM-CCS 1993*, ACM, pp. 62–73.
5. M. Bellare, H. Shi and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA 2005*, Springer LNCS 3376, 136–153, 2005.
6. O. Blazy, G. Fuchsbauer, M. Izabach'ene, A. Jambert, H. Sibert and D. Vergnaud. Batch Groth-Sahai. In *ACNS 2010*, Springer LNCS 6123, 218–235, 2010.

7. D. Boneh and X. Boyen. Short Signatures Without Random Oracles. In *EUROCRYPT 2004*, Springer LNCS 3027, 56–73, 2004.
8. R. Bobba, O. Fatemieh, F. Khan, C.A. Gunter and H. Khurana. Using Attribute-Based Access Control to Enable Attribute-Based Messaging. In *ACSAC 2006*, IEEE Computer Society 3027, 403–413, 2006.
9. D. Boneh, X. Boyen and H. Shacham. Short Group Signatures. In *CRYPTO 2004*, Springer LNCS 3152, 227–242, 2004.
10. X. Boyen. Mesh Signatures. In *EUROCRYPT 2007*, Springer LNCS 4515, 210–227, 2007.
11. J. Camenisch and A. Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *EUROCRYPT 2001*, Springer LNCS 2045, 93–118, 2001.
12. A. De Santis, G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *FOCS 1992*, 427–436, 1992.
13. A. Escala, J. Herranz and P. Morillo. Revocable Attribute-Based Signatures with Adaptive Security in the Standard Model. In *AFRICACRYPT 2011*, Springer LNCS 6737, 224–241, 2011.
14. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification. and signature problems. In *CRYPTO 1986*, Springer LNCS 263, 186–194, 1986.
15. K.B. Frikken, J. Li and M.J. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *NDSS 2006*, The Internet Society, 157–172, 2006.
16. G. Fuchsbauer. Automorphic Signatures in Bilinear Groups and an Application to Round-Optimal Blind Signatures. In *Cryptology ePrint Archive, Report 2009/320*, `http://eprint.iacr.org/2009/320.pdf`.
17. G. Fuchsbauer. Commuting signatures and verifiable encryption. In *EUROCRYPT 2011*, Springer LNCS 6632, 224–245, 2011.
18. Ma. Gagné, S. Narayan and R. Safavi-Naini. Short Pairing-Efficient Threshold-Attribute-Based Signature. In *Pairing 2012*, Springer LNCS 7708, 295–313, 2012.
19. S. Galbraith, K. Paterson and N.P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, **156**, 3113–3121, 2008.
20. E. Ghadafi, N.P. Smart and B. Warinschi. Practical zero-knowledge proofs for circuit evaluation. In *Coding and Cryptography: IMACC 2009*, Springer LNCS 5921, 469–494, 2009.
21. E. Ghadafi, N.P. Smart and B. Warinschi. Groth-Sahai proofs revisited. In *PKC 2010*, Springer LNCS 6056, 177–192, 2010.
22. V. Goyal, O. Pandey, A. Sahai, B. Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *CCS 2006*, ACM ,89–98 , 2006.
23. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT 2006*, Springer LNCS 4284, 444–459, 2006.
24. J. Groth. Fully anonymous group signatures without random oracles. In *ASIACRYPT 2007*, Springer LNCS 4833, 164–180, 2007.
25. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In SIAM Journal on Computing, volume 41(5), 1193–1232, 2012.
26. J. Herranz, F. Laguillaumie, B. Libert, and C. Ráfols. Short Attribute-Based Signatures for Threshold Predicates. In *CT-RSA 2012*, Springer LNCS 7178, 51–67, 2012.
27. M. Karchmer and A. Wigderson. On span programs. In *8th IEEE Structure in Complexity Theory*, 102–111, 1993.
28. S.A. Kakvi. Efficient fully anonymous group signatures based on the Groth group signature scheme. Masters thesis, University College London, 2010. `http://www5.rz.rub.de:8032/mam/foc/content/publ/thesis_kakvi10.pdf`.
29. D. Khader. Attribute Based Group Signatures with Revocation. In *Cryptology ePrint Archive, Report 2007/241*, `http://eprint.iacr.org/2007/241.pdf`.
30. D. Khader, L. Chen, J. H. Davenport. Certificate-Free Attribute Authentication. In *Cryptography and Coding: IMACC 2009*, Springer LNCS 5921, 301–325, 2009.
31. E. Kiltz. Chosen-Ciphertext Security from Tag-Based Encryption. In *TCC 2006*, Springer LNCS 3876, 581–600, 2006.
32. J. Li, M. H. Au, W. Susilo, D. Xie and K. Ren. Attribute-based signature and its applications. In *ASIACCS '10*, ACM, 60-69, 2010.
33. J. Li and K. Kim. Attribute-Based Ring Signatures. In *Cryptology ePrint Archive, Report 2008/394*, `http://eprint.iacr.org/2008/394.pdf`.
34. P. MacKenzie, M.K. Reiter and K. Yang. Alternatives to Non-malleability: Definitions, Constructions, and Applications. In *TCC 2004*, Springer LNCS 2951, 171–190, 2004.
35. H.K. Maji, M. Prabhakaran and M. Rosulek. Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resistance. In *Cryptology ePrint Archive, Report 2008/328*, `http://eprint.iacr.org/2008/328.pdf`.
36. H.K. Maji, M. Prabhakaran and M. Rosulek. Attribute-Based Signatures. In *CT-RSA 2011*, Springer LNCS 6558, 376–392, 2011.
37. M. Naor. On cryptographic assumptions and challenges. In *CRYPTO 2003*, Springer LNCS 2729, 96–109, 2003.
38. T. Okamoto and K. Takashima. Efficient Attribute-Based Signatures for Non-Monotone Predicates in the Standard Model. In *PKC 2011*, Springer LNCS 6571, 35–52, ,2011.
39. T. Okamoto and K. Takashima. Decentralized Attribute-Based Signatures. In *PKC 2013*, Springer LNCS 7778, 125–142, 2012.

40. A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. In *FOCS 1999*, 543–553, 1999.
41. A. Sahai, B. Waters. Fuzzy Identity-Based Encryption In *EUROCRYPT 2005*, Springer LNCS 3494, 457–473, 2005.
42. S. F. Shahandashti, and R. Safavi-Naini. Threshold Attribute-Based Signatures and Their Application to Anonymous Credential Systems. In *AFRICACRYPT 2009*, Springer LNCS 5580, 198–216, 2009.

A From Many Challenge Queries to a Single Query

Here we use a hybrid argument to show a reduction from an adversary $\mathcal{B}$ against the anonymity definition which makes at most $n(\lambda)$ calls (for some polynomial $n(\cdot)$) to the CH_b oracle to an adversary $\mathcal{F}$ which makes a single call to the CH_b oracle. The hybrid argument utilizes the fact that the adversary is allowed to learn the secret keys of any signer and hence it is capable of signing on behalf of any of them.

Lemma 1. *Let* DTABS *be a decentralized traceable attribute-based signature scheme. For any polynomial-time adversary $\mathcal{B}$ attacking the anonymity of* DTABS *with at most $n(\lambda)$ CH_b calls, there exists an adversary $\mathcal{F}$ attacking the anonymity of* DTABS *with only a single call to CH_b.*

Proof. We define a series of games $\{\mathsf{G}_j\}_{j=0}^{n(\lambda)}$, where in game G_j, the first j-th challenge queries by $\mathcal{B}$ are answered using the signing key $\{\mathsf{sk}_{\mathsf{sid}_0,a}\}_{a\in\mathcal{A}_0}$, whereas the rest of the queries are answered using $\{\mathsf{sk}_{\mathsf{sid}_1,a}\}_{a\in\mathcal{A}_1}$. Let P_j be the probability that adversary $\mathcal{B}$ wins game G_j. We have that

$$P_{n(\lambda)} = \Pr[\mathsf{Exp}^{\text{Anon-0}}_{\mathsf{DTABS},\mathcal{B}}(\lambda) = 1]$$
$$P_0 = \Pr[\mathsf{Exp}^{\text{Anon-1}}_{\mathsf{DTABS},\mathcal{B}}(\lambda) = 1]$$

In answering $\mathcal{B}$'s challenge queries, $\mathcal{F}$ randomly chooses $i \leftarrow [1, n(\lambda)]$, all challenge queries $j < i$ are answered using the key $\{\mathsf{sk}_{\mathsf{sid}_0,a}\}_{a\in\mathcal{A}_0}$. The i-th challenge query is answered using $\mathcal{F}$'s own challenge oracle and the remaining challenge queries are answered using the key $\{\mathsf{sk}_{\mathsf{sid}_1,a}\}_{a\in\mathcal{A}_1}$. The rest of $\mathcal{B}$'s queries are answered normally using the oracles available to $\mathcal{F}$. In the game, adversary $\mathcal{F}$ keeps an independent list $\mathsf{CL}_{\mathcal{F}}$ in which it records all its answers to $\mathcal{B}$'s challenge queries to ensure that $\mathcal{B}$ never asks to trace any of the challenge signatures. Thus, every time $\mathcal{B}$ asks a Trace query, $\mathcal{F}$ first looks up (m, σ, Ψ) in the list $\mathsf{CL}_{\mathcal{F}}$ and returns $\perp$ if it exists. Otherwise, it forwards such a request to its own Trace oracle and returns the answer to $\mathcal{B}$.

Adversar $\mathcal{F}$ is shown in Fig. 9, whereas the challenge and Trace oracles used by $\mathcal{F}$ to answer $\mathcal{B}$'s challenge and trace queries are given in Fig. 10.

Adversary $\mathcal{F}(\mathsf{pp} : \mathsf{AddS}(\cdot,\cdot), \mathsf{AddA}(\cdot), \mathsf{CrptA}(\cdot,\cdot), \mathsf{RevealS}(\cdot,\cdot), \mathsf{RevealA}(\cdot), \mathsf{CH}_b((\cdot,\cdot),(\cdot,\cdot),\cdot,\cdot), \mathsf{Trace}(\cdot,\cdot,\cdot))$

– $\mathsf{CL}_{\mathcal{F}} := \emptyset$.
– $\mathsf{cnt} := 0$.
– $i \leftarrow [1, n(\lambda)]$.
– $b^* \leftarrow \mathcal{B}(\mathsf{pp} : \mathsf{AddS}(\cdot,\cdot), \mathsf{AddA}(\cdot), \mathsf{CrptA}(\cdot,\cdot), \mathsf{RevealS}(\cdot,\cdot), \mathsf{RevealA}(\cdot), \mathsf{CH}((\cdot,\cdot),(\cdot,\cdot),\cdot,\cdot), \mathsf{Trace}(\cdot,\cdot,\cdot))$.
– If $\mathsf{cnt} < i$ Then $\sigma \leftarrow \mathsf{CH}_b((0,\perp),(0,\perp),\epsilon,\perp)$.[6] \\ Oracle Query
– Return b^*.

Fig. 9. Adversary $\mathcal{F}$.

[6] This statement is to ensure that $\mathcal{F}$ will always make one call to its challenge oracle.

$\underline{\mathsf{CH}((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)}$:
- If $\Psi(\mathcal{A}_0) \neq 1$ or $\Psi(\mathcal{A}_1) \neq 1$ Then Return $\perp$.
- $\mathsf{cnt} := \mathsf{cnt} + 1$.
- If $\mathsf{cnt} = i$ Then
  - $\sigma \leftarrow \mathsf{CH}_b((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)$. \\ Oracle Query
- Else
  - $\{\mathsf{sk}_{\mathsf{sid}_0,a}\}_{a\in\mathcal{A}_0} \leftarrow \mathsf{RevealS}(\mathsf{sid}_0, \mathcal{A}_0)$. \\ Oracle Query
  - $\{\mathsf{sk}_{\mathsf{sid}_1,a}\}_{a\in\mathcal{A}_1} \leftarrow \mathsf{RevealS}(\mathsf{sid}_1, \mathcal{A}_1)$. \\ Oracle Query
  - If $\mathsf{cnt} < i$ Then
    - $\sigma \leftarrow \mathsf{Sign}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\mathcal{A}_0}, \{\mathsf{sk}_{\mathsf{sid}_0,a}\}_{a\in\mathcal{A}_0}, m, \Psi)$.
  - Else
    - $\sigma \leftarrow \mathsf{Sign}(\{\mathsf{aavk}_{\mathsf{aid}(a)}\}_{a\in\mathcal{A}_1}, \{\mathsf{sk}_{\mathsf{sid}_1,a}\}_{a\in\mathcal{A}_1}, m, \Psi)$.
- $\mathsf{CL}_{\mathcal{F}} := \mathsf{CL}_{\mathcal{F}} \cup \{(m, \sigma, \Psi)\}$.

$\underline{\mathsf{Trace}(m, \sigma, \Psi)}$:
- If $(m, \sigma, \Psi) \in \mathsf{CL}_{\mathcal{F}}$ Then Return $\perp$.
- Return $\mathsf{Trace}(m, \sigma, \Psi)$. \\ Oracle Query

Fig. 10. The challenge oracle (left) and the Trace oracle (right) used by adversary $\mathcal{F}$.

Since i is chosen uniformly at random from the set $[1, n(\lambda)]$, we have for every $j \in [1, n(\lambda)]$ that

$$\Pr[\mathsf{Exp}^{\text{Anon-0}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1 | i = j] = P_j$$
$$\Pr[\mathsf{Exp}^{\text{Anon-1}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1 | i = j] = P_{j-1}$$

Thus, we have

$$\begin{aligned}\Pr[\mathsf{Exp}^{\text{Anon-0}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1] &= \sum_{j=1}^{n(\lambda)} \Pr[\mathsf{Exp}^{\text{Anon-0}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1 | i = j] \\ &= \sum_{j=1}^{n(\lambda)} P_j \cdot \frac{1}{n(\lambda)}\end{aligned}$$

Similarly, we have

$$\begin{aligned}\Pr[\mathsf{Exp}^{\text{Anon-1}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1] &= \sum_{j=1}^{n(\lambda)} \Pr[\mathsf{Exp}^{\text{Anon-1}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1 | i = j] \\ &= \sum_{j=1}^{n(\lambda)} P_{j-1} \cdot \frac{1}{n(\lambda)}\end{aligned}$$

By the above two equations, we have

$$\begin{aligned}\mathsf{Adv}^{\text{Anon}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\text{Anon-0}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{Anon-1}}_{\mathsf{DTABS},\mathcal{F}}(\lambda) = 1]\right| \\ &= \frac{1}{n(\lambda)} \cdot \left|P_{n(\lambda)} - P_0\right| \\ &= \frac{1}{n(\lambda)} \cdot \mathsf{Adv}^{\text{Anon}}_{\mathsf{DTABS},\mathcal{B}}(\lambda)\end{aligned}$$

B Properties of Simulation-Sound Non-Interactive Zero-Knowledge Proofs

The properties we require from a simulation-sound non-interctive zero-knowledge proof system are:

- **(Perfect) Completeness:** $\forall\lambda \in \mathbb{N}$, $\forall(x,w) \in \mathcal{R}$, we have

$$\Pr\left[(\mathsf{crs},\mathsf{xk}) \leftarrow \mathsf{Setup}(1^\lambda); \pi \leftarrow \mathsf{Prove}(\mathsf{crs},x,w) : \mathsf{Verify}(\mathsf{crs},x,\pi)=1\right] = 1 \ .$$

- **Soundness:** $\forall\lambda \in \mathbb{N}$, $\forall x \notin \mathcal{L}$, we have for all adversaries $\mathcal{F}$

$$\Pr\left[(\mathsf{crs},\mathsf{xk}) \leftarrow \mathsf{Setup}(1^\lambda); \pi \leftarrow \mathcal{F}(\mathsf{crs},x) : \mathsf{Verify}(\mathsf{crs},x,\pi)=1\right] \leq 2^{-\lambda} \ .$$

  If the above probability is 0, we say the system has *perfect soundness*.
- **Knowledge Extraction:** A proof system is a *Proof of Knowledge* or has *Knowledge Extraction* if there exists an efficient extractor algorithm $\mathsf{Extract}$ which can extract the witness from any proof the adversary outputs. Note that if a proof system is a proof of knowledge then it is sound. More formally, for all adversaries $\mathcal{F}$, we have

$$\Pr\Big[(\mathsf{crs},\mathsf{xk}) \leftarrow \mathsf{Setup}(1^\lambda); (x,\pi) \leftarrow \mathcal{F}(\mathsf{crs}); w \leftarrow \mathsf{Extract}(\mathsf{crs},\mathsf{xk},x,\pi) \\ : \mathsf{Verify}(\mathsf{crs},x,\pi)=0 \text{ OR } (x,w)\in\mathcal{R}\Big] \leq 1-\nu(\lambda) \ .$$

  If the above probability is 1, we say the system has *perfect knowledge extraction*.
- **Witness Indistinguishability:** The system is *witness indistinguishable* if for all PPT adversaries $\mathcal{F}$, we have

$$\Pr\left[\begin{array}{l}(\mathsf{crs},\mathsf{xk}) \leftarrow \mathsf{Setup}(1^\lambda); (\mathsf{st}_{\mathsf{find}},x,w_0,w_1) \leftarrow \mathcal{F}_{\mathsf{find}}(\mathsf{crs}); b \leftarrow \{0,1\}; \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs},x,w_b); b^* \leftarrow \mathcal{F}_{\mathsf{guess}}(\mathsf{st}_{\mathsf{find}},\pi) \\ : (x,w_0)\in\mathcal{R} \ \wedge \ (x,w_1)\in\mathcal{R} \ \wedge \ b=b^*\end{array}\right] = \frac{1}{2}+\nu(\lambda) \ .$$

  If $\nu(\lambda)=0$, we say the system has *perfect witness indistinguishability*.
- **Zero-Knowledge:** The system is *zero-knowledge* if $\forall(x,w)\in\mathcal{R}$, we have for all PPT adversaries $\mathcal{F}$

$$\Pr\left[(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr}) \leftarrow \mathsf{SimSetup}(1^\lambda) : \mathcal{F}^{\mathsf{Sim}(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr},\cdot,\cdot)}(\mathsf{crs}_{\mathrm{Sim}})=1\right] \\ \approx \Pr\left[(\mathsf{crs},\mathsf{xk}) \leftarrow \mathsf{Setup}(1^\lambda) : \mathcal{F}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs})=1\right],$$

  where $\mathsf{Sim}(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr},x,w)$ outputs $\mathsf{SimProve}(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr},x)$ if $(x,w)\in\mathcal{R}$ or $\perp$ otherwise.
- **Simulation-Soundness:** The system is *simulation-sound* [40] if the adversary cannot produce a proof for a false statement even after seeing simulated proofs for possibly false statements. Formally, for all PPT adversaries $\mathcal{F}$ we have

$$Pr\left[\begin{array}{l}(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr}) \leftarrow \mathsf{SimSetup}(1^\lambda); (\pi^*,x^*) \leftarrow \mathcal{F}^{\mathsf{SimProve}(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr},\cdot)}(\mathsf{crs}_{\mathrm{Sim}}) \\ : (\pi^*,x^*)\notin\mathcal{Q} \ \wedge \ \mathsf{Verify}(\mathsf{crs},x,\pi)=1 \ \wedge \ x\notin\mathcal{L}\end{array}\right] \approx 0 \ .$$

  If we limit the number of queries to one, we call the system *one-time simulation-sound*.
- **Simulation Sound Extractability:** By combining simulation-soundness and knowledge extraction, we get *Simulation-Sound Extractable Proofs* [23]. This requires that we can extract a witness from any proof the adversary outputs even after seeing simulated proofs. More formally, (here we abuse the notation and assume that $\mathsf{SimSetup}$ now also outputs the extraction key xk), we have for all PPT adversaries $\mathcal{F}$ that

$$\Pr\Big[(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr},\mathsf{xk}) \leftarrow \mathsf{SimSetup}(1^\lambda); (x,\pi) \leftarrow \mathcal{F}^{\mathsf{SimProve}(\mathsf{crs}_{\mathrm{Sim}},\mathsf{tr},\cdot)}(\mathsf{crs},\mathsf{xk}); w \leftarrow \mathsf{Extract}(\mathsf{crs},\mathsf{xk},x,\pi) \\ : (x,\pi)\notin\mathcal{Q} \ \wedge \ \mathsf{Verify}(\mathsf{crs},x,\pi)=1 \ \wedge \ (x,w)\notin\mathcal{R}\Big] \leq \nu(\lambda) \ .$$

C Proof of Theorem 1

Proof. Correctness of the construction follows from that of the underlying building blocks.

Lemma 2. *If the NIZK proof system* NIZK_1 *is simulation-sound and zero-knowledge,* NIZK_2 *is zero-knowledge, the encryption scheme* PKE *is IND-CCA2 secure, and the hash function* $\mathcal{H}$ *is collision-resistance then the generic construction is fully anonymous (against full-key exposure).*

Proof. We show that if there exists an adversary $\mathcal{B}$ which breaks the anonymity of the construction, we can construct adversaries $\mathcal{F}_1$ against the NIZK property of the proof system NIZK_1, $\mathcal{F}_2$ against the NIZK property of the proof system NIZK_2, $\mathcal{F}_{3,d}$ for $d \in \{0,1\}$ against the IND-CCA2 security of the encryption scheme PKE, $\mathcal{F}_4$ against the simulation-soundness of NIZK_1, and $\mathcal{F}_5$ against the collision-resistance of the hash function $\mathcal{H}$ such that

$$\begin{aligned}\mathsf{Adv}^{\text{Anon}}_{\mathsf{DTABS},\mathcal{B}}(\lambda) \leq & 2 \cdot (\mathsf{Adv}^{\text{NIZK}}_{\mathsf{NIZK}_1,\mathcal{F}_1}(\lambda) + \mathsf{Adv}^{\text{NIZK}}_{\mathsf{NIZK}_2,\mathcal{F}_2}(\lambda)) + \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{PKE},\mathcal{F}_{3,0}}(\lambda) + \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{PKE},\mathcal{F}_{3,1}}(\lambda) \\ & + \mathsf{Adv}^{\text{SS}}_{\mathsf{NIZK}_1,\mathcal{F}_4}(\lambda) + \mathsf{Adv}^{\text{Coll}}_{\mathcal{H},\mathcal{F}_5}(\lambda).\end{aligned}$$

By the collision-resistance of $\mathcal{H}$, $\mathcal{B}$ has a negligible advantage in finding pairs $(\Psi^*, m^*) \neq (\Psi, m)$ such that $\mathcal{H}(\Psi^*, m^*, C) = \mathcal{H}(\Psi, m, C)$ for some ciphertext C. If this is not the case, we can use $\mathcal{B}$ to construct an adversary $\mathcal{F}_5$ which breaks the collision-resistance of the hash function $\mathcal{H}$. Thus, from now on we assume that there are no hash collisions.

- **Adversary $\mathcal{F}_1$:** Adversary $\mathcal{F}_1$ runs the Setup algorithm normally and chooses all the keys itself. The only two differences here is that the CRS crs_1 used for NIZK_1 is obtained from $\mathcal{F}_1$'s environment, whereas crs_2 used for NIZK_2 is chosen by $\mathcal{F}_1$ by running $(\mathsf{crs}_2, \mathsf{tr}_2) \leftarrow \mathsf{NIZK}_2.\mathsf{SimSetup}(1^\lambda)$. $\mathcal{F}_1$ forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$ to $\mathcal{B}$.
  When asked AddA queries, $\mathcal{F}_1$ chooses the secret/verification keys for the authorities itself. Thus, $\mathcal{F}_1$ can answer any AddS queries itself.
  To answer Trace queries, $\mathcal{F}_1$ just decrypts the ciphertext within the signature, and simulates the proof π_{Trace}.
  When asked a $\mathsf{CH}_b((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)$ query, $\mathcal{F}_1$ randomly chooses $b \leftarrow \{0,1\}$ and generates the signature by signer sid_b and then forwards the details of the witness for the signature to its environment which responsds with a proof π which $\mathcal{F}_1$ needs to tell if it is a real proof or a simulated one. $\mathcal{F}_1$ constructs the rest of the signature and forwards it to $\mathcal{B}$. The output of $\mathcal{F}_1$ is that of $\mathcal{B}$.
- **Adversary $\mathcal{F}_2$:** Adversary $\mathcal{F}_2$ runs the Setup algorithm normally and chooses all the keys itself. The only difference here is the the CRS crs_2 used for NIZK_2 is obtained from $\mathcal{F}_2$'s environment.
  Adversary $\mathcal{F}_2$ forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$ to $\mathcal{B}$.
  When asked AddA queries, $\mathcal{F}_2$ chooses the secret/verification keys for the authorities itself. Thus, $\mathcal{F}_2$ can answer any AddS queries itself.
  To answer Trace queries, $\mathcal{F}_2$ just decrypts the ciphertext within the signature, and the proof π_{Trace} is obtained from $\mathcal{F}_2$'s Prove oracle. To answer the challenge query $\mathsf{CH}_b((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)$, $\mathcal{F}_2$ randomly chooses $b \leftarrow \{0,1\}$ and generates the signature by signer sid_b and constructs the proof π by itself. It then forwards the challenge signature to $\mathcal{B}$. The output of $\mathcal{F}_2$ is whatever $\mathcal{B}$ outputs.
- **Adversaries $\mathcal{F}_{3,0}$ and $\mathcal{F}_{3,1}$:** The details of adversaries $\mathcal{F}_{3,0}$ and $\mathcal{F}_{3,1}$ against the IND-CCA security of PKE are identical except for the difference in answering $\mathcal{B}$'s challenge query.
  Adversary $\mathcal{F}_{3,d}$ gets epk from its IND-CCA game (therefore it does not know the corresponding decryption key esk) and chooses crs_1 for NIZK_1 and crs_2 for NIZK_2 as simulated strings. In its IND-CCA game, $\mathcal{F}_{3,d}$ has access to a decryption oracle Dec.

Adversary $\mathcal{F}_{3,d}$ starts $\mathcal{B}$ with input $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$. When asked AddA queries, $\mathcal{F}_{3,d}$ chooses the secret/verification keys for the authorities itself. Thus, $\mathcal{F}_{3,d}$ can answer any AddS/Sign queries itself. To answer Trace queries, $\mathcal{F}_{3,d}$ sends the ciphertext C used in the input signature to its decryption oracle and then simulates the proof π_{Trace} (since it does not know the tracing key).
When asked a $\mathsf{CH}_b((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)$ query, $\mathcal{F}_{3,d}$ randomly chooses $d \leftarrow \{0,1\}$ and sets $p_d := \mathsf{sid}_d$ and $p_{1-d} := 0^{|\mathsf{sid}_d|}$. It then uses (p_0, p_1) as the challenge pair in its IND-CCA game. When it receives the challenge ciphertext C, $\mathcal{F}_{3,d}$ constructs the rest of the challenge signature by simulating the proof π.
The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.
If in the game, $\mathcal{B}$ queries the Trace oracle on a signature (π', C), i.e. one that re-uses the challenge ciphertext C but the associated proof is different from that used in the challenge signature, then $\mathcal{F}_{3,d}$ outputs its guess d; otherwise, it outputs whatever $\mathcal{B}$ outputs.

- **Adversary $\mathcal{F}_4$:** Adversary $\mathcal{F}_4$ runs the Setup algorithm normally and chooses all the keys itself. The only two differences here is that the CRS crs_1 used for NIZK_1 is obtained from $\mathcal{F}_1$'s simulation-soundness environment and crs_2 for NIZK_2 is chosen by running $(\mathsf{crs}_2, \mathsf{tr}_2) \leftarrow \mathsf{NIZK}_2.\mathsf{SimSetup}(1^\lambda)$. $\mathcal{F}_4$ forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$ to $\mathcal{B}$.
When asked AddA queries, $\mathcal{F}_4$ chooses the secret/verification keys for the authorities itself. Thus, $\mathcal{F}_4$ can answer any AddS queries itself.
To answer Trace queries, $\mathcal{F}_4$ just decrypts the ciphertext within the signature, and simulates the proof π_{Trace}.
When asked a $\mathsf{CH}_b((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)$ query, $\mathcal{F}_4$ computes C as the encryption of the string of zeros of length equal to the bit length of the signer identity space and uses the simulated proof it gets from its game to construct the challenge signature that it forwards to $\mathcal{B}$.
If during the game, $\mathcal{B}$ managed to query the Trace oracle on a signature $\sigma' = (\pi', C)$, i.e. one that re-uses the same challenge ciphertext but associated with a different proof π', $\mathcal{F}_4$ outputs the statement with which the proof π' is associated along with π' as its answer in its simulation-soundness game. Otherwise, it aborts.

Lemma 3. *The generic construction is fully unforgeable if the NIZK proof systems* NIZK_1 *and* NIZK_2 *are sound, the hash function* $\mathcal{H}$ *(used in encoding pseudo-attributes) is collision-resistant, and the digital signature scheme* DS *and the tagged signature scheme* TS *are both existentially unforgeable.*

Proof. Since the NIZK proof systems NIZK_1 and NIZK_2 are sound, the adversary has a negligible advantage in breaking full unforgeability by faking proofs for a false statement. Also, by the security of the hash function $\mathcal{H}$, the adversary has a negligible probability in finding collision between the encoding of different pairs of message/signing predicate. Thus, we proceed to show that if there exists an adversary that wins the full unforgeability game then we can construct adversaries $\mathcal{F}_1$ against the unforgeability of the tagged signature scheme TS, and adversary $\mathcal{F}_2$ against the unforgeability of the digital signature scheme DS such that

$$\mathsf{Adv}^{\text{F-Unforge}}_{\mathsf{DTABS},\mathcal{B}}(\lambda) \leq \kappa(\lambda) \cdot \mathsf{Adv}^{\text{Unfor}}_{\mathsf{TS},\mathcal{F}_1}(\lambda) + \mathsf{Adv}^{\text{Unfor}}_{\mathsf{DS},\mathcal{F}_2}(\lambda),$$

where $\kappa(\lambda)$ is a polynomial in λ representing an upper bound on the number of honest attribute authorities $\mathcal{B}$ is allowed to use in the game.

- **Adversay $\mathcal{F}_1$:** Adversary $\mathcal{F}_1$ gets the tagged signature scheme's verification key vk from its game and has access to an oracle Sign that it uses to obtain tagged signatures that verify w.r.t. vk on messages and tags (i.e. identities and attributes) of its choice. Adversary $\mathcal{F}_1$ starts by running $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow$

$\mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$, $\mathsf{crs}_2 \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$ and creating $(\mathsf{tsk}, \mathsf{tvk})$ honestly. It also creates the key pair $(\mathsf{esk}, \mathsf{epk})$ for the encryption scheme. It then forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$ and $\mathsf{tk} := \mathsf{esk}$ to $\mathcal{B}$.

Adversary $\mathcal{F}_1$ randomly chooses $i \leftarrow \{1, \ldots, \kappa(\lambda)\}$ and guesses that $\mathcal{B}$'s forgery will involve forging an attribute managed by the attribute authority i. When asked AddA queries, for all authorities $j \neq i$, $\mathcal{F}_1$ chooses the secret/verification keys for the authority itself. For authority i, it sets its verification key to vk it got from its game (and thus it does not know the corresponding secret key). If in the game, $\mathcal{B}$ issues RevealA query on authority i, $\mathcal{F}_1$ aborts the game.

Whenever $\mathcal{B}$ asks AddS queries, if the user has attributes managed by authority i, it forwards such a query to its Sign oracle; otherwise, it answers the query itself by using the authorities' secret keys available to it. When asked for Sign queries on $(\mathsf{sid}, \mathcal{A}, m, \Psi)$, $\mathcal{F}_1$ answers the query by first encrypting the identity sid and producing a signature on the pseudo-attribute that verifies w.r.t tvk. Note that $\mathcal{F}_1$ knows tsk and hence it can produce such a signature. By the witness-indistinguishability of NIZK_1 (implied by the zero-knowledge property) $\mathcal{B}$ cannot tell how the modified predicate was satisfied and hence cannot distinguish this signature from a real signature where the actual attributes of the user are used. The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.

Eventually, when $\mathcal{B}$ outputs its forgery, $\mathcal{F}_1$ uses the NIZK_1's extraction key xk_1 to extract the witness and returns the tagged signature on the identity and the attribute if $\mathcal{B}$'s forgery involved forging a tagged signature. Otherwise, it aborts (i.e. if the forgery was by forging a pseudo-attribute). If the signature does not involve forged attributes managed by authority i that $\mathcal{F}_1$ has gussed, it aborts. The probability that $\mathcal{F}_1$ guesses the correct authority is $\frac{1}{\kappa(\lambda)}$.

- **Adversary $\mathcal{F}_2$:** By the collision-resistant property of the hash function $\mathcal{H}$, the adversary cannot find collisions between different predicate/message/ciphertext tuples and hence we ignore this case.

  Adversary $\mathcal{F}_2$ gets tvk from its game and has access to an oracle Sign that it uses to obtain digital signatures that verify w.r.t. tvk on messages of its choice. It runs $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$, $\mathsf{crs}_2 \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$. It also creates the key pair $(\mathsf{esk}, \mathsf{epk})$ for the encryption scheme. It then forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$ and $\mathsf{tk} := \mathsf{esk}$ to $\mathcal{B}$.

  When asked for AddA queries, $\mathcal{F}_2$ creates the authority keys itself. Whenever $\mathcal{B}$ asks AddS queries, $\mathcal{F}_2$ uses the corresponding authorities' secret keys $\mathsf{aask}_{\mathsf{aid}(a)}$ to create the key for the signer. When asked for Sign queries on $(\mathsf{sid}, \mathcal{A}, m, \Psi)$, $\mathcal{F}_2$ first encrypts sid and queries its Sign oracle to obtain a signature on the corresponding pseudo-attribute matching (Ψ, m, C) and then constructs the rest of the signature by generating the proof π. Again, by the witness-indistinguishability of NIZK_1 (implied by the zero-knowledge property) $\mathcal{B}$ cannot tell which witness was used in the proof. The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.

  Eventually, when $\mathcal{B}$ outputs its forgery, $\mathcal{F}_2$ uses NIZK_1's extraction key xk_1 to extract the witness and returns the signature on the pseudo-attribute $a_{\hat{\Psi}^*, m^*, C^*}$ if the forgery was done by forging a signature on a pseudo-attribute; otherwise, it aborts.

Lemma 4. *The generic construction is traceable if the NIZK proof system* NIZK_1 *is sound, and the digital signature scheme* DS *and the tagged signature scheme* TS *are both existentially unforgeable.*

Proof. The details are very similar to that of full unforgeability. The difference between the full unforgeability proof and the traceability proof is that here the adversary is not allowed to corrupt or learn the secret key of any attribute authority; otherwise, it is easy to create untraceable signatures.

Since the NIZK proof system NIZK_1 is sound, the adversary has a negligible advantage in succeeding by faking proofs for false statements. Thus, we proceed to show that if there exists an adversary that wins

the traceability game then we can construct adversaries $\mathcal{F}_1$ against the unforgeability of the tagged signature scheme TS, and adversary $\mathcal{F}_2$ against the unforgeability of the digital signature scheme DS such that

$$\mathsf{Adv}^{\text{Trace}}_{\mathsf{DTABS},\mathcal{B}}(\lambda) \leq \kappa(\lambda) \cdot \mathsf{Adv}^{\text{Unfor}}_{\mathsf{TS},\mathcal{F}_1}(\lambda) + \mathsf{Adv}^{\text{Unfor}}_{\mathsf{DS},\mathcal{F}_2}(\lambda),$$

where $\kappa(\lambda)$ is a polynomial in λ representing an upper bound on the number of honest attribute authorities $\mathcal{B}$ is allowed to use in the game.

- **Adversay $\mathcal{F}_1$:** Adversary $\mathcal{F}_1$ gets the tagged signature scheme's verification key vk from its game and has access to an oracle Sign that it uses to obtain tagged signatures that verify w.r.t. vk on messages and tags (i.e. identities and attributes) of its choice. Adversary $\mathcal{F}_1$ starts by running $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$, $\mathsf{crs}_2 \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$ and creating $(\mathsf{tsk}, \mathsf{tvk})$ honestly. It also creates the key pair $(\mathsf{esk}, \mathsf{epk})$ for the encryption scheme. It then forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$ and $\mathsf{tk} := \mathsf{esk}$ to $\mathcal{B}$.
  Adversary $\mathcal{F}_1$ randomly chooses $i \leftarrow \{1, \ldots, \kappa(\lambda)\}$ and guesses that $\mathcal{B}$'s untraceable signature involves the attribute authority i. When asked AddA queries, for all authorities $j \neq i$, $\mathcal{F}_1$ chooses the secret/verification keys for the authority itself. For authority i, it sets its verification key to vk it got from its game (and thus it does not know the corresponding secret key).
  Whenever $\mathcal{B}$ asks AddS queries, if the user has attributes managed by authority i, it forwards such a query to its Sign oracle; otherwise, it answers the query itself by using the authorities' secret keys available to it. When asked for Sign queries on $(\mathsf{sid}, \mathcal{A}, m, \Psi)$, $\mathcal{F}_1$ answers the query by producing a signature on the pseudo-attribute that verifies w.r.t tvk. Note that $\mathcal{F}_1$ knows tsk and hence it can produce such a signature. By the witness-indistinguishability of NIZK_1 (implied by the zero-knowledge property) $\mathcal{B}$ cannot tell how the modified predicate was satisfied and hence cannot distinguish this signature from a real signature where the actual attributes of the user are used. The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.
  Eventually, when $\mathcal{B}$ outputs its forgery, $\mathcal{F}_1$ uses the NIZK_1's extraction key xk_1 to extract the witness and returns the tagged signature on the identity and the attributes if the forgery involves the attribute authority i. Otherwise, it aborts. The probability that $\mathcal{F}_1$ guesses the correct authority is $\frac{1}{\kappa(\lambda)}$.
- **Adversary $\mathcal{F}_2$:** Adversary $\mathcal{F}_2$ gets tvk from its game and has access to an oracle Sign that it uses to obtain signatures that verify w.r.t. tvk on messages of its choice. It runs $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$ and $\mathsf{crs}_2 \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$. It also creates the key pair $(\mathsf{esk}, \mathsf{epk})$ for the encryption scheme. It then forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H})$ and $\mathsf{tk} := \mathsf{esk}$ to $\mathcal{B}$.
  When asked for AddA queries, $\mathcal{F}_2$ creates the authority keys itself. Whenever $\mathcal{B}$ asks AddS queries, $\mathcal{F}_2$ uses the corresponding authorities' secret keys $\mathsf{aask}_{\mathsf{aid}(a)}$ to create the key for the signer. When asked for Sign queries on $(\mathsf{sid}, \mathcal{A}, m, \Psi)$, $\mathcal{F}_2$ queries its Sign oracle to obtain a signature on the corresponding pseudo-attribute matching (Ψ, m, C) and constructs the rest of the signature by encrypting sid and generating the proof π using sid and the signature on the pseudo-attribute $a_{\Psi,m,C}$ as a witness. By the witness indistinguishability of NIZK_1, the adversary cannot tell which witness was used in creating the proof. The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.
  Eventually, when $\mathcal{B}$ outputs its forgery, $\mathcal{F}_2$ uses NIZK_1's extraction key xk_1 to extract the witness and returns the signature on the pseudo-attribute $a_{\hat{\Psi}^*,m^*,C^*}$ if $\mathcal{B}$ created the untraceable signature by using a forged signature on a new pseudo-attribute that it did not get from querying $\mathcal{F}_2$; otherwise, $\mathcal{F}_2$ aborts.

D Proof of Theorem 2

Proof. Correctness of the construction follows from that of the underlying building blocks.

Lemma 5. *If the* NIZK_1 *and* NIZK_2 *proof systems are zero-knowledge, the tag-based encryption scheme* TPKE *is selective-tag weakly IND-CCA secure, the one-time signature* OTS *is strongly existentially unforgeable, and the hash functions* $\hat{\mathcal{H}}$ *and* $\mathcal{H}$ *are collision-resistant then the construction is fully anonymous (against full-key exposure).*

Proof. We show that if there exists an adversary $\mathcal{B}$ which breaks the anonymity of the construction, we can construct adversaries $\mathcal{F}_1$ against the collision-resistance of the hash function $\hat{\mathcal{H}}$, $\mathcal{F}_2$ against the strong unforgeability of the one-time signature OTS, $\mathcal{F}_3$ against the collision-resistance of the hash function $\mathcal{H}$, $\mathcal{F}_4$ against the NIZK property of the proof system NIZK_1, $\mathcal{F}_5$ against the NIZK property of the proof system NIZK_2, and $\mathcal{F}_6$ against the selective-tag weakly IND-CCA security of the tag-based encryption scheme TPKE.

By the collision-resistance of the hash function $\hat{\mathcal{H}}$, $\mathcal{B}$ has a negligible probability in finding otsvk' such that $\hat{\mathcal{H}}(\mathsf{otsvk}')$ collides with the tag $\hat{\mathcal{H}}(\mathsf{otsvk}^*)$ we will use for the tag-based ciphertext within the challenge signature. If this is not the case, then we can use $\mathcal{B}$ to construct an adversary $\mathcal{F}_1$ that breaks the collision-resistance of $\hat{\mathcal{H}}$.

By the strong existential unforgeability of OTS, we also have that $\mathcal{B}$ has a negligible probability in forging a one-time signature under otsvk^* used in the challenge signature. If this is not the case, we can construct an adversary $\mathcal{F}_2$ that wins the strong unforgeability game of the one-time signature.

By the collision-resistance of the $\mathcal{H}$, $\mathcal{B}$ has a negligible advantage in finding pairs $(\Psi^*, m^*) \neq (\Psi, m)$ such that $\mathcal{H}(\Psi^*, m^*, C_{\text{tbe}}, \hat{\mathcal{H}}(\mathsf{otsvk})) = \mathcal{H}(\Psi, m, C_{\text{tbe}}, \hat{\mathcal{H}}(\mathsf{otsvk}))$. If this is not the case, we can use $\mathcal{B}$ to construct an adversary $\mathcal{F}_3$ which breaks the collision-resistance of the hash function $\mathcal{H}$. Thus, from now on we assume that there are no such hash collisions.

We now start NIZK_1 in the simulation setting which is by the security of NIZK_1 is indistinguishable from the soundness setting. The proof π is thus now zero-knowledge and hence does not reveal any information about the witness.

We now also start NIZK_2 in the simulation setting which is indistinguishable from the soundness setting. The proof π_{Trace} is now also zero-knowledge and hence $\mathcal{B}$ cannot tell simulated proofs from real proofs.

We now proceed to show how to use $\mathcal{B}$ to construct an adversary $\mathcal{F}_6$ against the selective-tag weakly IND-CCA security of TPKE.

Adversary $\mathcal{F}_6$ runs the Setup algorithm where it starts by randomly choosing a key pair $(\mathsf{otsvk}^*, \mathsf{otssk}^*)$ for OTS that it will use in answering $\mathcal{B}$'s challenge signature. Note that we needed to choose the key pair beforehand as the tag-based encryption scheme is only selective-tag secure and hence the challenger in the ST-WIND-CCA game needs to know the challenge tag before it sends the public-key epk for TPKE. $\mathcal{F}_6$ sends $\hat{\mathcal{H}}(\mathsf{otsvk}^*)$ to its challenger and gets back epk. In its game, $\mathcal{F}_6$ has access to a decryption oracle Dec which it can query on any ciphertext under any tag different from $\hat{\mathcal{H}}(\mathsf{otsvk}^*)$. $\mathcal{F}_6$ chooses crs_1 and crs_2 as simulation reference strings. $\mathcal{F}_6$ also chooses a key pair $(\mathsf{tvk}, \mathsf{tsk})$ for the digital signature scheme DS. $\mathcal{F}_6$ forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H}, \hat{\mathcal{H}})$ to $\mathcal{B}$.

When asked AddA queries, $\mathcal{F}_6$ chooses the secret/verification keys for the authorities itself. Thus, $\mathcal{F}_6$ can answer any AddS queries itself.

To answer the challenge query $\mathsf{CH}_b((\mathsf{sid}_0, \mathcal{A}_0), (\mathsf{sid}_1, \mathcal{A}_1), m, \Psi)$, $\mathcal{F}_6$ sends $(\mathsf{sid}_0, \mathsf{sid}_1)$ as its challenge in its ST-WIND-CCA game and gets a ciphertext under the tag $\hat{\mathcal{H}}(\mathsf{otsvk}^*)$ of either the plaintext sid_0 or sid_1 which he needs to distinguish. $\mathcal{F}_6$ can now construct the rest of the challenge signature by simulating the proof π and signing the whole thing with otssk^* to obtain σ_{ots}.

To answer Trace queries, $\mathcal{F}_6$ just uses its decryption oracle to get the decryption of C_{tbe} which is part of the signature and then simulates proof π_{Trace}. Note that since we have chosen the challenge tag otsvk^* uniformly at random and since we already eliminated any case where any signature sent to Trace uses the

same tag as that we used for the challenge signature, such a query will be accepted by $\mathcal{F}_6$'s decryption oracle because the tag is different from the tag used in the challenge ciphertext. The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.

Finally, when $\mathcal{B}$ outputs its guess, $\mathcal{F}_6$'s output is that of $\mathcal{B}$

Lemma 6. *The construction is fully unforgeable if* NIZK_1 *and* NIZK_2 *proof systems are sound, the hash functions* $\mathcal{H}$ *(used in encoding pseudo-attributes) and* $\hat{\mathcal{H}}$ *are collision-resistant, and the one-time signature* OTS*, the digital signature* DS *and the tagged signature* TS *are all existentially unforgeable.*

Proof. We instantiate both NIZK_1 and NIZK_2 proof systems in the soundness setting and hence the adversary cannot break full unforgeability by faking proofs for a false statement. Thus, we proceed to show that if there exists an adversary that wins the full unforgeability game then we can construct adversaries $\mathcal{F}_1$ against the unforgeability of the tagged signature scheme TS, adversary $\mathcal{F}_2$ against the unforgeability of the digital signature scheme DS, adversary $\mathcal{F}_3$ against the strong unforgeability of the one-time signature scheme OTS, and adversaries $\mathcal{F}_4$ and $\mathcal{F}_5$ against the collision-resistance of the hash functions $\mathcal{H}$ and $\hat{\mathcal{H}}$, respectively, such that

$$\mathsf{Adv}^{\text{F-Unforge}}_{\mathsf{DTABS},\mathcal{B}}(\lambda) \leq \kappa(\lambda) \cdot \mathsf{Adv}^{\text{Unfor}}_{\mathsf{TS},\mathcal{F}_1}(\lambda) + \mathsf{Adv}^{\text{Unfor}}_{\mathsf{DS},\mathcal{F}_2}(\lambda) + \delta(\lambda) \cdot \mathsf{Adv}^{\text{Unfor}}_{\mathsf{OTS},\mathcal{F}_3}(\lambda) + \mathsf{Adv}^{\text{Coll}}_{\mathcal{H},\mathcal{F}_4}(\lambda) + \mathsf{Adv}^{\text{Coll}}_{\hat{\mathcal{H}},\mathcal{F}_5}(\lambda),$$

where $\kappa(\lambda)$ and $\delta(\lambda)$ are polynomials in λ representing an upper bound on the number of honest attribute authorities and sign queries, respectively, $\mathcal{B}$ is allowed to make in the game.

By the security of the hash function $\mathcal{H}$, $\mathcal{B}$ has a negligible probability in finding collisions between the encodings of different tuples of signing predicate/message/ciphertext/OTS verification key. If this is not the case, we can use $\mathcal{B}$ to construct an adversary $\mathcal{F}_4$ that breaks the collision-resistance of $\mathcal{H}$.

Similarly, by the collision-resistance of the hash function $\hat{\mathcal{H}}$, $\mathcal{B}$ has a negligible probability in finding two different one-time signature keys $\mathsf{otsvk} \neq \mathsf{otsvk}'$ such that $\hat{\mathcal{H}}(\mathsf{otsvk}) = \hat{\mathcal{H}}(\mathsf{otsvk}')$. If this is not the case, we can use $\mathcal{B}$ to construct an adversary $\mathcal{F}_5$ that breaks the collision-resistance of $\hat{\mathcal{H}}$.

Thus, from now on we assume that there are no hash collisions.

- **Adversay $\mathcal{F}_1$:** Adversary $\mathcal{F}_1$ gets the tagged signature scheme's verification key vk from its game and has access to an oracle Sign that it uses to obtain tagged signatures that verify w.r.t. vk on messages and tags (i.e. identities and attributes) of its choice. Adversary $\mathcal{F}_1$ starts by running $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$, $(\mathsf{crs}_2, \mathsf{xk}_2) \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$ and choosing $(\mathsf{tsk}, \mathsf{tvk})$ honestly. It also creates the key pair $(\mathsf{esk}, \mathsf{epk})$ for the tag-based encryption scheme TPKE. It then forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H}, \hat{\mathcal{H}})$ and $\mathsf{tk} := \mathsf{esk}$ to $\mathcal{B}$.
  Adversary $\mathcal{F}_1$ randomly chooses $i \leftarrow \{1, \ldots, \kappa(\lambda)\}$ and guesses that $\mathcal{B}$'s forgery will involve forging an attribute managed by the attribute authority i. When asked AddA queries, for all authorities $j \neq i$, $\mathcal{F}_1$ chooses the secret/verification keys for the authority itself. For authority i, it sets its verification key to vk it got from its game (and thus it does not know the corresponding secret key). If in the game, $\mathcal{B}$ issues $\mathsf{RevealA}$ query on authority i, $\mathcal{F}_1$ aborts the game.
  Whenever $\mathcal{B}$ asks AddS queries, if the user has attributes managed by authority i, it forwards such a query to its Sign oracle; Otherwise, it answers the query itself by using the authorities' secret keys available to it.
  When asked for Sign queries on $(\mathsf{sid}, \mathcal{A}, m, \Psi)$, $\mathcal{F}_1$ first chooses a fresh key pair $(\mathsf{otsvk}, \mathsf{otssk})$ for the one-time signature OTS and encrypts sid using $\hat{\mathcal{H}}(\mathsf{otsvk})$ as a tag. It then uses tsk to generate a signature on the pseudo-attribute. Note that by the witness-indistinguishability of NIZK_1, $\mathcal{B}$ cannot tell whether we constructed the signature by using real attributes that sid possesses or using a pseudo-attribute. $\mathcal{F}_1$ forwards the signature to $\mathcal{B}$. The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.

Eventually, when $\mathcal{B}$ outputs its forgery, $\mathcal{F}_1$ uses the NIZK_1's extraction key xk_1 to extract the witness and returns the tagged signature on the identity and the attribute if $\mathcal{B}$'s forgery involved forging a tagged signature. Otherwise, it aborts. $\mathcal{F}_1$ also aborts if the forgery does not involve forged attributes managed by authority i that $\mathcal{F}_1$ has guessed. The probability that $\mathcal{F}_1$ guesses the correct authority is $\frac{1}{\kappa(\lambda)}$.
By the existential unforgeability of the tagged signature scheme, the probability of $\mathcal{B}$ winning in this case is negligible.

- **Adversary $\mathcal{F}_2$:** Adversary $\mathcal{F}_2$ gets tvk from its game and has access to an oracle Sign that it uses to obtain signatures that verify w.r.t. tvk on messages of its choice. It runs $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$, $(\mathsf{crs}_2, \mathsf{xk}_2) \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$. It also creates the key pair $(\mathsf{esk}, \mathsf{epk})$ for the tag-based encryption scheme TPKE. It then forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H}, \hat{\mathcal{H}})$ and $\mathsf{tk} := \mathsf{esk}$ to $\mathcal{B}$.
When asked for AddA queries, $\mathcal{F}_2$ creates the authority keys itself.
Whenever $\mathcal{B}$ asks AddS queries, $\mathcal{F}_2$ uses the corresponding authorities' secret keys $\mathsf{aask}_{\mathsf{aid}(a)}$ to create the key for the signer.
When asked for Sign queries on $(\mathsf{sid}, \mathcal{A}, m, \Psi)$, $\mathcal{F}_2$ generates a fresh key pair $(\mathsf{otsvk}, \mathsf{otssk})$ for the one-time signature. It then produces the tag-based ciphertext C_{tbe} which is the encryption of sid using $\hat{\mathcal{H}}(\mathsf{otsvk})$ as a tag and forwards the pseudo-attribute $\mathcal{H}(\Psi, m, C_{\text{tbe}}, \hat{\mathcal{H}}(\mathsf{otsvk}))$ to its oracle. After receiving the signature from its own sign oracle, $\mathcal{F}_2$ produces the proof π using sid and the signature on the pseudo-attribute as a witness. It then signs the whole thing with otssk and returns $\sigma := (\sigma_{\mathsf{ots}}, \pi, C_{\text{tbe}}, \mathsf{otsvk})$ as the answer. By the witness-indistinguishability of the proof system NIZK_1, the adversary cannot tell which witness was used in the proof. The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.
Eventually, when $\mathcal{B}$ outputs its forgery, $\mathcal{F}_2$ uses NIZK_1's extraction key xk_1 to extract the witness and returns the signature on the pseudo-attribute $a_{m^*, \hat{\Psi}^*, C_{\text{tbe}}, \hat{\mathcal{H}}(\mathsf{otsvk}^*)}$ if the forgery was done by forging a signature on a pseudo-attribute; otherwise, it aborts.
By the existential unforgeability of the of the digital signature scheme DS, the probability of $\mathcal{B}$ winning in this case is negligible.
- **Adversary $\mathcal{F}_3$:** Adversary $\mathcal{F}_3$ gets otsvk^* from its game and has access to an oracle Sign that it uses to obtain a single one-time signature that verify w.r.t. otsvk on a message of its choice. It runs $(\mathsf{crs}_1, \mathsf{xk}_1) \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$, $(\mathsf{crs}_2, \mathsf{xk}_2) \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$. It also creates the key pair $(\mathsf{esk}, \mathsf{epk})$ for the tag-based encryption scheme TPKE and $(\mathsf{tsk}, \mathsf{tvk})$ for the digital signature DS. It then forwards $\mathsf{pp} := (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{epk}, \mathsf{tvk}, \mathbb{A}, \mathcal{H}, \hat{\mathcal{H}})$ and $\mathsf{tk} := \mathsf{esk}$ to $\mathcal{B}$.
When asked for AddA queries, $\mathcal{F}_3$ creates the authority keys itself. Whenever $\mathcal{B}$ asks AddS queries, $\mathcal{F}_3$ uses the corresponding authorities' secret keys $\mathsf{aask}_{\mathsf{aid}(a)}$ to create the key for the signer.
Adversary $\mathcal{F}_3$ randomly chooses $i \leftarrow \{1, \ldots, \delta(\lambda)\}$ and guesses that $\mathcal{B}$'s forgery will involve forging a one-time signature that verifies under otsvk^* used in answering the i-th signing query.
When asked for the j-th Sign query on $(\mathsf{sid}, \mathcal{A}, m, \Psi)$, if $j \neq i$, $\mathcal{F}_3$ chooses a fresh key pair $(\mathsf{otsvk}, \mathsf{otssk})$ for the one-time signature scheme and answers the query by itself. If $j = i$, $\mathcal{F}_3$ encrypts sid using $\hat{\mathcal{H}}(\mathsf{otsvk}^*)$ (i.e. the public key it got from its game) as a tag to obtain C_{tbe}, it then generates a signature on the pseudo-attribute and constructs the proof π. It then forwards $(\pi, C_{\text{tbe}}, \mathsf{otsvk}^*)$ as the message to its one-time signature signing oracle to get a one-time signature σ_{ots}. $\mathcal{F}_3$ sends the signature $\sigma := (\sigma_{\mathsf{ots}}, \pi, C_{\text{tbe}}, \mathsf{otsvk}^*)$ to $\mathcal{B}$.
The rest of $\mathcal{B}$'s queries are answered normally as in Fig. 1.
Eventually, when $\mathcal{B}$ outputs its forgery, $\mathcal{F}_3$ aborts if the $\mathcal{B}$'s forgery did not involve forging a one-time signature that verifies w.r.t otsvk^* it got from its game. The probability that $\mathcal{B}$ forges a one-time signature that verifies w.r.t the same otsvk^* is $\frac{1}{\delta(\lambda)}$.

By the strong existential unforgeability of the one-time signature OTS, $\mathcal{B}$ has a negligible advantage in wining this case.
This concluded the proof.

Lemma 7. *The construction is traceable if the* NIZK_1 *proof system is sound, and the digital signature* DS *and the tagged signature* TS *are all existentially unforgeable.*

Proof. Since the NIZK proof system NIZK_1 is sound, the adversary has a negligible advantage in succeeding by faking proofs for false statements. Thus, we proceed to show that if there exists an adversary that wins the traceability game then we can construct adversaries $\mathcal{F}_1$ attacking the unforgeability of the tagged signature scheme TS, and adversary $\mathcal{F}_2$ attacking the unforgeability of the digital signature scheme DS such that

$$\mathsf{Adv}^{\text{Trace}}_{\mathsf{DTABS},\mathcal{B}}(\lambda) \leq \kappa(\lambda) \cdot \mathsf{Adv}^{\text{Unfor}}_{\mathsf{TS},\mathcal{F}_1}(\lambda) + \mathsf{Adv}^{\text{Unfor}}_{\mathsf{DS},\mathcal{F}_2}(\lambda),$$

where $\kappa(\lambda)$ is a polynomial in λ representing an upper bound on the number of honest attribute authorities $\mathcal{B}$ is allowed to use in the game.

The reductions are very similar to those in the full unforgeability game. The difference here is that $\mathcal{B}$ is not allowed to make any CrptA or $\mathsf{RevealA}$ queries.