Noname manuscript No.
(will be inserted by the editor)

Scalable Attribute-based Group Key Establishment: from Passive to Active and Deniable

Rainer Steinwandt · **Adriana Suárez Corona**

Received: date / Accepted: date

Abstract A protocol compiler is presented which transforms any unauthenticated (attribute-based) group key establishment protocol into an authenticated attribute-based group key establishment. If the protocol to which the compiler is applied does not make use of long-term secrets, then the resulting protocol is, in addition, deniable. In particular, applying our compiler to an unauthenticated 2-round protocol going back to Burmester and Desmedt results in a 3-round solution for attribute-based group key establishment, offering both forward secrecy and deniability.

Keywords Group key establishment · Deniabilility · Attribute-based key encapsulation · Attribute-based signature

1 Introduction

Attribute-based group key establishment addresses a situation where a common session key needs to be established among a group of users, who are in possession of certain credentials. In such an attribute-based setting it is not considered vital who the exact communication partners are, but rather that they possess an appropriate set of attributes. As an example, in a university setting one could think of a session key for a virtual meeting among department chairs or, more fine-grained, department chairs within a particular set of departments of the university. Similarly, in an enterprise environment, one can think of a scenario where a session key should be available to anyone who is a member of a particular project team (possibly crossing company boundaries).

RS was supported by the Spanish "Ministerio de Economía y Competitividad" through the project grant MTM-2012-15167.
ASC was supported by project MTM2010 - 18370 - C04- 01 and FPU grant AP2007-03141, cofinanced by the European Social Fund.

R. Steinwandt
Florida Atlantic University, USA
E-mail: rsteinwa@fau.edu

A. Suárez Corona
University of Denver, USA
E-mail: Adriana.SuarezCorona@du.edu

A two-party variant of this problem has been studied by Wang et al. [**?**,**?**,**?**], for instance, and also Camenisch et al.'s work on *Credential-Authenticated Key Exchange* [**?**] can be mentioned in this line of work. In the group setting, Steinwandt and Suárez Corona proposed a two-round solution [**?**] in the random oracle model, and a generic one-round solution has been proposed by Gorantla et al. [**?**]. The latter does not provide forward secrecy, however. To construct a solution offering forward secrecy, [**?**] suggests to consider a modification of the Katz-Yung compiler from CRYPTO 2003 [**?**] using an attribute-based signature scheme, and apply such to an (unauthenticated) protocol by Burmester and Desmedt [**?**]. Working out the details of such an approach is left as future work, however.

A central technical tool in the Katz-Yung compiler from CRYPTO 2003 is a strongly unforgeable signature scheme which is used to authenticate all protocol messages. From a privacy point of view, this technique is problematic, if one also aims at deniability: signatures in a protocol transcript may serve as evidence that a particular user participated in a protocol execution, and it is not clear how such a protocol transcript can be simulated (as is commonly imposed in formal definitions of deniability). In the usual public-key setting, proposals for deniable group key exchange have, e. g., been put forward by Bohli and Steinwandt [**?**], Zhang et al. [**?**], and Chen et al. [**?**]. More recently, a modification of Katz and Yung's compiler has been proposed, which replaces the use of a strongly unforgeable signature scheme through a combination of a strongly unforgeable message authentication code, a ring signature, and a multi key encapsulation [**?**]. The latter compiler allows to transform any unauthenticated group key establishment protocol that does not make use of long-term secrets into an authenticated group key establishment that is also deniable. Our definition for deniability of an attribute-based group key establishment builds on [**?**].

Contribution In this paper we introduce a modification of the Katz-Yung compiler, which transforms any (unauthenticated attribute-based) group key establishment into an authenticated attribute-based group key establishment. If the unauthenticated protocol does not make use of long-term secrets the resulting protocol is deniable. In particular, applying our compiler to a two-round protocol going back to Burmester and Desmedt [**?**,**?**] results in an attribute-based group key establishment with three rounds, offering deniability as well as forward secrecy. To the best of our knowledge, this is the first general construction to derive attribute-based group key establishments from unauthenticated solutions. Also, we are not aware of any prior general constructions to derive deniable group key establishment protocols in the attribute-based context. Interestingly, not even a formal definition of deniability seems to have been given in this context so far, which is somewhat surprising: from the security point of view, there is no obvious reason for an attribute-based signature scheme to hide the exact set of attributes that is used to produce a signature, and it is not a given that an authenticated attribute-based key establishment protocol is deniable in an intuitive sense.

2 Preliminaries

As main technical tools, we will make use of an attribute-based key encapsulation along with a suitable symmetric encryption, which serves as data encapsulation mechanism, to send an identical message to multiple protocol participants in a confidential manner. For implementing authentication we also make use of a message authentication code and an attribute-based signature. Throughout this paper, the security parameter will be denoted by k, and notions like negligible or polynomial time refer to k.

2.1 Attribute-based key encapsulation and symmetric encryption

In analogy to an ordinary key encapsulation mechanism (KEM), attribute-based key encapsulation allows the creation of an encapsulation of a symmetric key such that only users who are part of a specified access structure can recover the encapsulated key. Gorantla et al. [**?**] formalized this notion as follows:

Definition 1 (Encapsulation-policy attribute-based KEM)
An *encapsulation-policy attribute-based KEM (EP-AB-KEM)* is a tuple of polynomial time algorithms $(\texttt{KSetup}, \texttt{KGen}, \texttt{KEncaps}, \texttt{KDecaps}, \texttt{KDelegate})$:

`KSetup` is probabilistic and run by a trusted authority: on input the security parameter 1^k and a universe of attributes $\mathscr{U}$, a master secret key mk and public system parameters pm are generated.

`KGen` is probabilistic and run by a trusted authority: on input the master secret key mk and a set of attributes $U \subseteq \mathscr{U}$ belonging to a user, a secret key dk_U for these attributes is generated.

`KEncaps` is probabilistic and run by a user. On input an access structure $\mathbb{A} \subseteq 2^{\mathscr{U}}$, this algorithm generates an encapsulation C and a symmetric key K which can only be recovered from C by users possessing attributes satisfying $\mathbb{A}$. We assume that the encapsulation implicitly contains $\mathbb{A}$.

`KDecaps` is a deterministic algorithm run by a user with a set of attributes $U \subseteq \mathscr{U}$. On input the encapsulation C containing the access structure $\mathbb{A}$ and dk_U, this algorithm outputs the symmetric key K, if U is contained in the access structure $\mathbb{A}$. Otherwise an error symbol $\perp$ is returned.

`KDelegate` is a probabilistic algorithm run by a user with a set of attributes $U \subseteq \mathscr{U}$. On input the private key dk_U and a set $\tilde{U} \subseteq U$, this algorithm outputs the private key $dk_{\tilde{U}}$ corresponding to $\tilde{U}$.

It is required that $\texttt{KDecaps}(C, \mathbb{A}, dk_U) = K$ if $(K, C) \leftarrow \texttt{KEncaps}(params, \mathbb{A})$ and $U \in \mathbb{A}$.

Similar to an indistinguishability notion for ciphertext policy attribute based encryption (CP-ABE), cf. Bethencourt et al. [**?**], a security model for EP-AB-KEM schemes is defined:

Definition 2 (IND-AB-CCA)
An encapsulation-policy attribute-based KEM is *semantically secure against chosen ciphertext attacks (IND-AB-CCA)* if the advantage of any probabilistic polynomial time adversary $\mathscr{A}$ in the game described in Figure **??** is negligible. Here the *advantage* of an adversary $\mathscr{A}$ is defined as

$$\mathrm{Adv}_{\mathscr{A}}^{\text{ind-ab-cca}}(k) := \left| 2 \cdot \Pr[b = b'] - 1 \right|.$$

To actually encrypt messages in our compiler, we combine an EP-AB-KEM with a symmetric encryption scheme that offers security in the real-or-random sense (cf. Bellare et al. [**?**]):

Definition 3 (Symmetric encryption scheme) A symmetric encryption scheme is a triple of polynomial time algorithms $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ as follows:

- `KeyGen` is probabilistic. Given the security parameter 1^k, it generates a secret key K.
- `Enc` is probabilistic. Given a secret key K and a message $m \in \{0,1\}^*$, this algorithm generates a ciphertext C.

- **Setup** The challenger $\mathscr{C}$ runs the `KSetup` algorithm and hands the resulting public parameters *params* to the adversary $\mathscr{A}$. The master secret key *mk* remains private.
- **Phase 1** The adversary is allowed to make the following queries (adaptively):
  - $\mathsf{Extract}(U)$: The challenger $\mathscr{C}$ responds by running the algorithm `KExtract` with input $(params, mk, U)$ and hands the resulting private key dk_U to $\mathscr{A}$.
  - $\mathsf{Decaps}(U, C)$: Notice that the encapsulation C implicitly contains an access structure $\mathbb{A}$. If U satisfies $\mathbb{A}$, the challenger $\mathscr{C}$ responds by running the algorithm `KExtract` to get the private key dk_U. Then it runs algorithm `KDecaps` on input (dk_U, C) and sends the resulting symmetric key to the adversary. If U does not satisfy $\mathbb{A}$, it returns an error symbol $\perp$.
- **Challenge** After $\mathscr{A}$ decides to end Phase 1, it outputs an access structure $\mathbb{A}^* \in 2^{\mathscr{U}}$. The challenger $\mathscr{C}$ chooses a random bit b and runs the `KEncaps` algorithm on input $\mathbb{A}^*$ to generate a pair (K_1, C^*). Then it selects uniformly at random from the key space a key K_0 and sends (K_b, C^*) as challenge to $\mathscr{A}$.
- **Phase 2** Identical to Phase 1.
- **Guess** The adversary outputs a value $b' \in \{0,1\}$ and wins if and only if $b = b'$ and the following restrictions on the queries hold:
  - No $\mathsf{Extract}$ query on input $U \in \mathbb{A}^*$ is made.
  - No Decaps query on input (C^*, U), such that $U \in \mathbb{A}^*$ is made.

Fig. 1 IND-AB-CCA security of an EP-AB-KEM scheme

- `Dec` is deterministic. Given a ciphertext C and a secret key K, this algorithm outputs either a message m or a dedicated error symbol $\perp$.

We require that for every key K the relation $m = \mathtt{Dec}_K(\mathtt{Enc}_K(m))$ holds for all $m \in \{0,1\}^*$.

To express security in the real-or-random sense, let $\mathscr{RR}(m,b)$ be a *real-or-random oracle*, i. e., on input $m \in \{0,1\}^*$ a query $\mathscr{RR}(m,1)$ returns the message m, whereas a query $\mathscr{RR}(m,0)$ returns a uniformly at random chosen bitstring of the same length as m. For a probabilistic polynomial time algorithm $\mathscr{A}$, now consider the experiment described in Figure **??** where $\mathscr{E}_K(\cdot)$ respectively $\mathscr{D}_K(\cdot)$ is an oracle which applies the encryption algorithm $\mathtt{Enc}_K(\cdot)$ respectively the decryption algorithm $\mathtt{Dec}_K(\cdot)$ to its input.

- **Setup:** A secret bit b is chosen uniformly at random and a secret key $K \leftarrow \mathtt{KeyGen}(1^k)$ is created.
- **Challenge:** The adversary $\mathscr{A}$ has unrestricted access to the 'composed oracle' $\mathscr{E}_K(\mathscr{RR}(\cdot,b))$. Further, $\mathscr{A}$ has access to $\mathscr{D}_K(\cdot)$ subject to the restriction that no ciphertexts must be queried to $\mathscr{D}_K(\cdot)$ that have been output by $\mathscr{E}_K(\mathscr{RR}(\cdot,b))$.
- **Guess:** The adversary outputs a bit $b' \in \{0,1\}$.

Fig. 2 ROR-CCA security of a symmetric encryption scheme

Building on this experiment, security in the real-or-random sense is defined by measuring $\mathscr{A}$'s success in correctly identifying the secret bit b used by the real-or-random oracle:

Definition 4 (ROR-CCA security)
A symmetric encryption scheme $(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ is *secure in the sense of real-or-random (ROR-CCA)*, if the advantage $\mathrm{Adv}^{\text{ror-cca}}_{\mathscr{A}} = \mathrm{Adv}^{\text{ror-cca}}_{\mathscr{A}}(k) :=$

$$\Pr\left[1 \leftarrow \mathscr{A}^{\mathscr{E}_K(\mathscr{RR}(\cdot,1)),\mathscr{D}_K(\cdot)}(1^k) : K \leftarrow \mathtt{KeyGen}(1^k)\right]$$
$$-\Pr\left[1 \leftarrow \mathscr{A}^{\mathscr{E}_K(\mathscr{RR}(\cdot,0)),\mathscr{D}_K(\cdot)}(1^k) : K \leftarrow \mathtt{KeyGen}(1^k)\right]$$

is negligible for all probabilistic polynomial time algorithms $\mathscr{A}$.

2.2 Message authentication codes and attribute-based signatures

To solve the problem of authentication without jeopardizing deniability, we use a message authentication code as well as a suitable attribute-based signature.

Definition 5 (Message authentication code)
A *message authentication code (MAC)* is a tuple $(\texttt{MKeyGen}, \texttt{Tag}, \texttt{Verify})$ of polynomial time algorithms as follows:

- $\texttt{MKeyGen}$ is probabilistic. Given the domain parameters $\mathbb{D}$, it generates a secret key K.
- $\texttt{Tag}$ is probabilistic. Given a message $m \in \{0,1\}^*$ and a secret key K it generates a message tag $\theta := \texttt{Tag}_K(m) \in \{0,1\}^*$ on m.
- $\texttt{Verify}$ is deterministic. Given a message m, a secret key K and a candidate tag θ, this algorithm returns 1 if θ is a valid tag for the message m and 0 otherwise.

The compiler described below assumes that the message authentication code we employ is strongly unforgeable under adaptive chosen message attacks (cf. [**?**]). Figure **??** describes the corresponding experiment, where $\mathscr{T}_K(\cdot)$ respectively $\mathscr{V}_K(\cdot)$ describes an oracle which applies $\texttt{Tag}_K(\cdot)$ respectively $\texttt{Verify}_K(\cdot)$ to its inputs.

Definition 6 (SUF-CMA security)
Denote by $(\texttt{MKeyGen}, \texttt{Tag}, \texttt{Verify})$ a message authentication code. It is said to be *strongly unforgeable under adaptive chosen message attacks/secure in the sense of SUF-CMA* if for all probabilistic polynomial time adversaries $\mathscr{A}$ the advantage

$$\mathrm{Adv}^{\text{suf-cma}}_{\mathscr{A}} = \mathrm{Adv}^{\text{suf-cma}}_{\mathscr{A}}(k) := \Pr[\mathsf{Succ}^{\text{suf-cma}}_{\mathscr{A}}]$$

is negligible. Here $\mathsf{Succ}^{\text{suf-cma}}_{\mathscr{A}}$ denotes the event that $\mathscr{A}$ wins the experiment in Figure **??**.

- **Setup:** A secret key $K \leftarrow \texttt{MKeyGen}(\mathbb{D})$ is created
- **Challenge:** The adversary $\mathscr{A}$ has unrestricted access to the tagging oracle $\mathscr{T}_K(\cdot)$ and the verification oracle $\mathscr{V}_K(\cdot)$.
- **Guess:** $\mathscr{A}$ outputs a (message, tag)-pair (m, θ) and wins if and only if $\texttt{MVer}_K(m, \theta) = 1$ and either m has never beeen queried to $\mathscr{T}_K(\cdot)$ or no query of the form $\mathscr{T}_K(m)$ returned the tag θ.

Fig. 3 SUF-CMA security of a message authentication code

Finally, to construct our compiler we make use of an attribute-based signature scheme. Signatures produced with these schemes are verified with an access structure. A signature is valid if the attributes of the signer satisfy it.

Definition 7 (Attribute-based signature scheme)
An *attribute-based signature scheme* is given by a tuple of polynomial time algorithms $(\texttt{STSetup}, \texttt{SSetup}, \texttt{SKeyGen}, \texttt{SSign}, \texttt{SVerify})$ as follows:

- $\texttt{STSetup}$ is probabilistic and run by a signature trustee: on input the security parameter 1^k and a universe of attributes $\mathscr{U}$, a public reference information *pm* is produced.
- $\texttt{SSetup}$ is probabilistic and run by a trusted authority: on input the public information *pm*, a master secret key *msk* and master public key *mpk* are generated.
- $\texttt{SKeyGen}$ is probabilistic and run by a trusted authority: on input the master secret key *msk* and a set of attributes U belonging to a user, a secret key sk_U for these attributes is generated.

- `SSign` is probabilistic and run by a user who wants to sign a message m with his secret key sk_U to be verified with an access structure $\mathbb{B}$: on input $m \in \mathscr{M}$, sk_U and $\mathbb{B}$, this algorithm generates a signature σ.
- `SVerify` is deterministic and run by a user who wants to verify if a signature has been created by a user with a set of attributes in the verification access structure $\mathbb{B}$: on input a message m, a signature σ and an access structure $\mathbb{B} \subseteq 2^{\mathscr{U}}$, this algorithm outputs TRUE if σ is a valid signature for m under sk_U for some $U \in \mathbb{B}$. Otherwise the algorithm outputs FALSE.

We require that for any access structure $\mathbb{B}$, any key sk_U produced by `SKeyGen`, where $U \in \mathbb{B}$, and for any message m the relation $\texttt{SVerify}(m, \texttt{SSign}_{sk_U}(m, \mathbb{B}), \mathbb{B}) = \text{TRUE}$ holds.

One feature we expect from an attribute-based signature is some kind of anonymity, in the sense that neither the adversary nor the trusted authority can know which attribute set satisfying the access structure was the one actually used to sign the message. A strong form of this design goal is known as *perfect privacy* [**?**]:

- **Setup:** The challenger $\mathscr{C}$ runs the algorithm `STSetup` and hands the public parameters to $\mathscr{A}$. Then $\mathscr{A}$ runs `SSetup` and sends (msk, mpk) to the challenger.
- **Challenge:** The adversary $\mathscr{A}$ outputs a message m^*, an access structure $\mathbb{A}^*$ and two attribute sets U_0, U_1 to $\mathscr{C}$, such that $U_0, U_1 \in \mathbb{A}^*$. The challenger $\mathscr{C}$ chooses a bit $b \in \{0,1\}$ uniformly at random, computes a signature $\texttt{SSign}_{sk_{U_b}}(m^*, \mathbb{A}^*)$ and hands it to $\mathscr{A}$.[a]
- **Guess:** The adversary outputs a guess b' for b and wins if and only if $b = b'$.

[a] Notice that the adversary has the master secret key. Therefore, in particular, $\mathscr{A}$ can recover valid secret keys of all the attribute sets.

Fig. 4 Privacy for attribute-based signatures

Definition 8 (Privacy) An attribute-based signature scheme is *computationally private* if the advantage of any probabilistic polynomial time adversary $\mathscr{A}$ in the game described in Figure **??** is negligible. Here the *advantage* of an adversary $\mathscr{A}$ is defined as

$$\mathrm{Adv}^{\text{absig-pr}}_{\mathscr{A}}(k) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

The attribute-based signature scheme is said to be *perfectly private* if the advantage is 0 for any adversary with possibly unbounded computational power.

Of course, as for other kinds of digital signatures, for an attribute-based signature scheme we also expect an appropriate form of existential unforgeability. More specifically, we impose the following.

Definition 9 (UF-CMAA security) An attribute-based signature scheme is *existentially unforgeable* against *chosen message and attribute set attacks* (*UF-CMAA*) if for any probabilistic polynomial time adversary $\mathscr{A}$ the advantage $\mathrm{Adv}^{\text{uf-cmaa}}_{\mathscr{A}}(k) := \Pr[\mathsf{Succ}^{\text{uf-cmaa}}_{\mathscr{A}}]$ in the game described in Figure **??** is negligible. Here $\mathsf{Succ}^{\text{uf-cmaa}}_{\mathscr{A}}$ denotes the event that $\mathscr{A}$ wins the experiment in Figure **??**.

Remark 1 In the definitions in [**?**], claim predicates are used instead of access structures. Given a universe of attributes $\mathscr{U}$, an access structure $\mathbb{B}$ is a set of sets of attributes ($\mathbb{B} \subseteq 2^{\mathscr{U}}$). A signature produced with the private key corresponding to a set of attributes U is valid

- **Setup:** The challenger $\mathscr{C}$ runs the algorithm $\texttt{STSetup}$ and $\texttt{SSetup}$ and hands the public parameters and mpk to $\mathscr{A}$.
- **Challenge:** The adversary is allowed to ask (adaptively) queries for:
  - private keys for attribute sets U. The challenger returns sk_U;
  - signatures of a signer with attribute set U on a message m. The challenger responds with $\texttt{SSign}_{sk_U}(m)$.
- **Forgery Phase:** The adversary outputs a tuple $(\mu, \sigma, \mathbb{A})$, where μ is a message and $\mathbb{A}$ is an access structure. The adversary wins if and only if all of the following hold:
  - $\texttt{SVer}(\mu, \sigma, \mathbb{A}) = \text{TRUE}$;
  - in the challenge phase, no secret key sk_U for a $U \in \mathbb{A}$ has been queried;
  - in the challenge phase, all inputs of signature queries satisfy $m \neq \mu$ or $U \notin \mathbb{A}$.

Fig. 5 UF-CMAA security of an attribute-based signature

for the verification access structure $\mathbb{B}$ if $U \in \mathbb{B}$. On the other hand, claim predicates are monotone boolean functions on $2^{\mathscr{U}}$. A signature produced with the set of attributes U is valid for the claim predicate γ if $\gamma(U) = 1$. Notice they are equivalent. Given an access structure $\mathbb{B}$ we can define a claim predicate $\gamma_{\mathbb{B}}$ such that $\gamma_{\mathbb{B}}(U) = 1$ if and only if $U \in \mathbb{B}$. Conversely, for a claim predicate γ, we can define the access structure $\mathbb{B}_\gamma = \{U \in 2^{\mathscr{U}} : \gamma(U) = 1\}$. We use access structures here, so that their use in the compiler becomes more clear. If only threshold policies are needed, a more efficient proposal by Herranz et al. is also available [**?**] where the security notion is slightly weaker: *selective-predicate and adaptive message unforgeability*. They propose also some extensions to their protocol when dealing with some other kinds of predicates.

3 Security model

By $\mathscr{U} \subseteq \{0,1\}^{\mathrm{O}(1)}$ we denote a non-empty constant size *universe of attributes*. To model the security of attribute-based key establishment, we adapt the model in [**?**] to an attribute-based setting, following to some extent [**?**]. It should be noted that one could attempt to strengthen the model below, which may offer a natural avenue for follow-up work. In particular, we do not address the question of group key *agreement* in the presence of malicious insiders, which (though not for attribute-based authentication) has been considered by Bohli [**?**], for instance.

3.1 Security goals: key secrecy and authentication

Protocol participants. We model the protocol participants as probabilistic polynomial time algorithms labelled with subsets U of the attribute universe $\mathscr{U}$. Depending on the specific protocol, users may have further long-term secrets. An identifier $U \in 2^{\mathscr{U}}$ represents any user whose attributes are exactly the ones specified in U, and we do not distinguish between users having the same attributes. Each user U can execute a polynomial number of protocol instances $\prod_U^s$ concurrently ($s \in \mathbb{N}$), and to each protocol instance $\prod_U^s$ we associate the following seven variables:

used_U^s: indicates if this instance is or has been involved in a protocol run;
state_U^s: keeps the state information during the execution;
term_U^s: indicates if the protocol execution has terminated. In that case, is set to TRUE;
pid_U^s: stores the sets of attributes a user U aims at establishing a key with: $\mathsf{pid}_U^s \subseteq 2^{\mathscr{U}}$ and we impose that $U \in \mathsf{pid}_U^s$;

sk_U^s: stores the session key once it is accepted by the instance $\prod_U^s$. Before, a distinguished NULL value is stored;
sid_U^s: stores a non-secret session identifier that can be used as public reference to the session key stored in sk_U^s;
acc_U^s: indicates if the session key stored in sk_U^s has been accepted, i. e., the protocol run was successful.

Initialization. Each user obtains his long-term secret key ak_U in a trusted initialization phase (without adversarial interference), where a trusted authority generates those keys for each $U \in 2^{\mathscr{U}}$ using a master secret key *msk*.

Remark 2 This initialization phase can become trivial when no long-term secrets are required, for instance, for some unauthenticated protocols.

Communication network and adversarial capabilities. We assume the network is non-private and fully asynchronous. Arbitrary point-to-point connections among the users are available, but no dedicated broadcast functionality is assumed to be in place. Thus, 'broadcasting' a message does neither guarantee that all intended recipients receive the same message, nor that all intended recipients receive a message at all. The adversary $\mathscr{A}$ is modeled as a probabilistic polynomial time algorithm with complete control over the communication network. Its capabilities are captured by the subsequent *oracles*:

$\mathsf{Send}(U_i, s_i, M)$: sends the message M to instance $\prod_{U_i}^{s_i}$ of user U_i and returns the protocol message output by that instance after receiving M. The Send oracle also enables $\mathscr{A}$ to initialize a protocol execution: sending the special message $M = (\mathbb{B}, b)$, containing an access structure and a role flag ($b \in \{\text{INIT}, \overline{\text{INIT}}\}$), to an unused instance $\prod_{U_i}^{s_i}$ initializes a protocol run among users U' such that each $U' \in \mathbb{B}$. After such a query, $\prod_{U_i}^{s_i}$ sets $\mathsf{pid}_{U_i}^{s_i} := \mathbb{B}$, $\mathsf{used}_{U_i}^{s_i} :=$ TRUE, and processes the first step of the protocol.
$\mathsf{Execute}(U_1, s_1, \ldots, U_r, s_r)$: returns a complete protocol transcript among the specified unused instances, setting $\mathsf{pid}_{U_1}^{s_1} = \ldots \mathsf{pid}_{U_r}^{s_r} = \{U_1, \ldots, U_r\}$.
$\mathsf{Reveal}(U, s)$: returns the session key sk_U^s if $\mathsf{acc}_U^s =$ TRUE and a NULL value otherwise.
$\mathsf{Corrupt}(U)$: for a user $U \in \mathscr{U}$ this query returns U's complete long-term secret key.

An *active* adversary has access to all of the above oracles: we can interpret such an adversary as being able to eavesdrop, delay, suppress and insert messages at will. Adversaries without access to the Send oracle are considered to be *passive*—we assume in their presence all protocol messages to be sent as specified in the protocol description. The adversary $\mathscr{A}$ has also access to a Test oracle, which can be queried only once: the query $\mathsf{Test}(U, s)$ can be made with an instance $\prod_U^s$ that has accepted a session key. In that case, a bit $b \leftarrow \{0, 1\}$ is chosen uniformly at random; for $b = 0$, the session key stored in sk_U^s is returned, and for $b = 1$ a uniformly at random chosen element from the space of session keys is returned.

To exclude useless protocols, we consider only *correct* attribute-based group key establishments [**?**]:

Definition 10 (Correctness) An attribute-based group key establishment is *correct* if a single protocol execution among users $\mathscr{V} \subseteq 2^{\mathscr{U}}$ involves instances $\prod_U^{s_U}$ ($U \in \mathscr{V}$) such that with overwhelming probability all of the following hold:

- all users in $\mathscr{V}$ accept, i. e., $\mathsf{acc}_U^{s_U} =$ TRUE for all $U \in \mathscr{V}$;
- all users in $\mathscr{V}$ obtain the same session identifier, i. e., $\mathsf{sid}_U^{s_U}$ is identical for all $U \in \mathscr{V}$;

- all users in $\mathscr{V}$ accept the same session key, i. e., sk_U^{sU} is identical and $\neq$NULL for all $U \in \mathscr{V}$;
- all communication partners are specified as desired communication partner, i. e., $\mathscr{V} \subseteq \mathsf{pid}_U^{sU}$ for all $U \in \mathscr{V}$.

We need to specify when an attack is valid. We define the following notion of *partnering*:

Definition 11 (Partnering) Two terminated instances $\prod_U^s$ and $\prod_{U'}^{s'}$ with $U \neq U'$ are *partnered* if $\mathsf{sid}_U^s = \mathsf{sid}_{U'}^{s'}$, $\mathsf{acc}_U^s = \mathsf{acc}_{U'}^{s'} = \text{TRUE}$ and $\mathsf{pid}_U^s = \mathsf{pid}_{U'}^{s'}$.

Remark 3 According to Definition **??**, different instances of the same user cannot be partnered. This implies that no deterministic protocol can be secure (cf. [**?**] for a discussion).

Having fixed the definition of partnering, we can now specify which instances an adversary can attack:

Definition 12 (Freshness with forward secrecy) An instance $\prod_U^s$ is said to be *fresh* if none of the following events has occured:

- For some $U' \in \mathsf{pid}_U^s$ a $\mathsf{Corrupt}(U')$ query was executed before a query of the form $\mathsf{Send}(U'', s'', *)$ has taken place where $U'' \in \mathsf{pid}_U^s$.
- The adversary $\mathscr{A}$ queried $\mathsf{Reveal}(U, s)$;
- The adversary $\mathscr{A}$ queried $\mathsf{Reveal}(U', s')$ with $\prod_U^s$ and $\prod_{U'}^{s'}$ being partnered.

If forward secrecy is not a concern, the following weaker definition of freshness can be considered [**?**], and the subsequent results hold with either definition of freshness:

Definition 13 (Freshness without forward secrecy) An instance $\prod_U^s$ is said to be *fresh* if none of the following events has occurred:

- the adversary queried $\mathsf{Corrupt}(U')$ for some $U' \in \mathsf{pid}_U^s$;
- the adversary queried $\mathsf{Reveal}(U, s)$;
- the adversary queried $\mathsf{Reveal}(U', s')$ for an instance $\prod_{U'}^{s'}$ that is partnered with $\prod_U^s$.

We write $\mathsf{Succ}_{\mathscr{A}}$ for the event that the adversary $\mathscr{A}$ queries Test with a fresh instance and correctly guesses the random bit b used by the Test oracle. The *advantage* of $\mathscr{A}$ in winning the game is

$$\mathrm{Adv}_{\mathscr{A}}^{\mathrm{ke}} = \mathrm{Adv}_{\mathscr{A}}^{\mathrm{ke}}(k) := \left| \Pr[\mathsf{Succ}] - \frac{1}{2} \right|.$$

Our compiler transforms an unauthenticated, secure attribute-based group key establishment protocol, i. e., a protocol where the adversary is assumed to be passive, into an authenticated one.

Definition 14 (Key secrecy) An attribute-based group key establishment protocol is *secure*, if $\mathrm{Adv}_{\mathscr{A}}^{\mathrm{ke}} = \mathrm{Adv}_{\mathscr{A}}^{\mathrm{ke}}(k)$ is negligible for all passive probabilistic polynomial time adversaries $\mathscr{A}$.

When considering active adversaries the following definition applies:

Definition 15 (Key secrecy & authentication) An attribute-based group key establishment protocol is said to be *authenticated and secure*, if $\mathrm{Adv}_{\mathscr{A}}^{\mathrm{ke}} = \mathrm{Adv}_{\mathscr{A}}^{\mathrm{ke}}(k)$ is negligible for all active probabilistic polynomial time adversaries $\mathscr{A}$.

3.2 Deniability in the attribute-based setting

A second design goal of our compiler is to obtain protocols that are not only authenticated, but at the same time offer deniability. Informally, this means that the involvement of a particular user (in this setting, a user possessing a particular set of attributes) in the protocol cannot be proved, even by the central authority. Here we adapt the formalization of deniability used in [**?**] (which in turn builds on [**?**]) to the attribute-based setting.

3.2.1 Deniability

Let $\mathscr{A}_d$ denote a probabilistic polynomial time algorithm with the security parameter 1^k and public information pk from the initialization phase as input. We distinguish two phases. In the first one, $\mathscr{A}_d$ has access to the Corrupt oracle only, and can (adaptively) corrupt an arbitrary subset of the users (including the case of no user or the authority, and therefore all, being corrupted). In a second phase, $\mathscr{A}_d$ interacts with the protocol participants via the Reveal- and Send oracle.[1] Neither Corrupt nor Test queries can be made in this phase. The adversary $\mathscr{A}_d$ then outputs a transcript $T_{\mathscr{A}_d} = T_{\mathscr{A}_d}(k, pk)$ to evidence the involvement of a particular user in the group key establishement. Let $T_{\mathscr{A}_d} = T_{\mathscr{A}_d}(k)$ be the random variable describing $T_{\mathscr{A}_d}(k, pk)$ with the randomness for $\mathscr{A}_d$, for all protocol instances, and in the initialization phase being chosen uniformly at random.

A second algorithm, the simulator $\mathscr{S}_d$, obtains the same input as $\mathscr{A}_d$, but cannot invoke uncorrupted users. This algorithm can only query the Corrupt oracle—no Execute, Reveal, Send, or Test queries are allowed. The simulator $\mathscr{S}_d$ also outputs a bitstring $T_{\mathscr{S}_d}(k, pk)$, and analogously as for $\mathscr{A}_d$ we define a random variable $T_{\mathscr{S}_d}(k)$ based on uniformly at random chosen randomness. Consider the following experiment for a probabilistic polynomial time distinguisher $\mathscr{X}$ outputting 0 or 1: the challenger flips a random coin $b \in \{0,1\}$ uniformly at random. If $b = 1$, the transcript $T_{\mathscr{A}_d}(k)$ is handed to $\mathscr{X}$, whereas for $b = 0$ the transcript $T_{\mathscr{S}_d}(k)$ is handed to $\mathscr{X}$. The distinguisher $\mathscr{X}$ wins whenever the guess b' it outputs for b is correct; the advantage of $\mathscr{X}$ is denoted by

$$\mathrm{Adv}^{\mathrm{den}}_{\mathscr{X}} := \left| \Pr[b = b'] - \frac{1}{2} \right| .$$

In the attribute-based setting we suggest the following definition of deniability:

Definition 16 (Deniability) An attribute-based group key establishment protocol is *deniable* if for every polynomial time adversary $\mathscr{A}_d$ as specified above there exists a probabilistic polynomial time simulator $\mathscr{S}_d$ such that the following two conditions hold:

- With overwhelming probability, the number of Corrupt queries of $\mathscr{S}_d$ is less than or equal to the number of Corrupt queries of $\mathscr{A}_d$.
- The simulator $\mathscr{S}_d$ has corrupted the central authority only if $\mathscr{A}_d$ has corrupted it.
- For each probabilistic polynomial time distinguisher $\mathscr{X}$, the advantage $\mathrm{Adv}^{\mathrm{den}}_{\mathscr{X}}$ in the above experiment is negligible.

The idea behind this definition is that any evidence that might be provided by $\mathscr{A}_d$ for the involvement of an uncorrupted user can as well be produced by a simulator $\mathscr{S}_d$ who does *not* involve this uncorrupted user. This alternative explanation of $\mathscr{A}_d$'s "evidence" enables an uncorrupted user to deny involvement in the protocol. For the alternative explanation to be plausible, the simulator may not involve more corrupted users than $\mathscr{A}_d$ does.

[1] The Send oracle can in particular be used to simulate Execute queries.

4 From unauthenticated to authenticated and deniable

For our compiler we assume that some IND-AB-CCA secure encapsulation-policy attribute-based key encapsulation (see Definition **??**) (`KKeyGen`, `KEncaps`, `KDecaps`) and a ROR-CCA secure symmetric encryption scheme (see Definition **??**) (`KeyGen`, `Enc`, `Dec`) are provided. Also, let (`MKeyGen`, `Tag`, `Verify`) be an SUF-CMA secure message authentication code (see Definition **??**) and (`STSetup`, `SSetup`, `SKeyGen`, `SSign`, `SVerify`) a UF-CMAA secure attribute-based signature scheme (see Definition **??**) which is computationally private in the sense of Definition **??**.

4.1 Description of the proposed compiler

If P is a (secure) *unauthenticated* attribute-based group key establishment protocol, then applying the compiler below yields an *authenticated* attribute-based protocol P', and in case P offers forward secrecy, then this property is preserved. If the original protocol P does not make use of long-term secrets, then the resulting protocol P' is in addition *deniable*. We assume that $U_0, \ldots, U_{n-1}$ are the users who want to establish a secret key, and we distinguish one party as being the initiator of the protocol. He will be the only one producing an attribute-based signature in the Round 0 message. Without loss of generality we assume that U_0 plays this role. Moreover, for the ease of notation, in our description, we do not explicitly refer to individual instances.

Let $m_{i,l}^{(j)}$ be the message sent by user U_i in the j-th round of protocol P to user U_l. Then, without loss of generality, we can assume that instead of sending the messages $m_{i,l}^{(j)}$ in protocol P, the user U_i broadcasts the combined message

$$m_{i,j} = U_i || j || \left(m_{i,1}^{(j)}, \ldots, m_{i,n-1}^{(j)} \right).$$

In particular, $m_{i,j}$ includes an (unprotected) identifier of the sender U_i of the message and of the round number. Each recipient U_l can recover $m_{i,l}^{(j)}$ in the obvious way, and this change does neither affect the security against passive adversaries nor the round complexity of P.

Initialization phase. In the initialization phase, all the keys needed for protocol P are created. For better readability, we avoid the use of subindex U_i and use just i instead. The public keys are made available to all users (and the adversary).

The suggested compiler modifies protocol P as follows:

Introduction of Round 0. In this new initial round, each user U_i ($i \neq 0$):

- chooses a random nonce $r_i \in \{0,1\}^k$.
- broadcasts $U_i||0||r_i$.

The initiator U_0 additionally does the following:

- run `MKeyGen` to generate a key K_0 for the message authentication code;
- produces an attribute-based signature $\sigma := \mathtt{SSign}_{sk_0}(K_0||\mathsf{pid}_0||U_0||0||r_0, \mathsf{pid}_0)$;
- computes $(K, C) \leftarrow \mathtt{KEncaps}(\mathsf{pid}_0)$;
- produces a ciphertext $E := \mathtt{Enc}_K(K_0||\mathsf{pid}_0||U_0||0||r_0||\sigma)$ using the symmetric key K;
- computes a tag $\mathtt{tag}_0 := \mathtt{Tag}_{K_0}(C, E)$, and

– broadcasts $U_0||0||(C,E)||\mathsf{tag}_0$.

After receiving the Round 0 message of all parties, each user U_i executes the following steps:

– set $\mathsf{nonces}_{U_i} := ((U_1,r_1),\cdots,(U_n,r_n))$ and store this value;
– run $\mathtt{KDecaps}_{dk_i}(C)$ to obtain K;
– decrypt the ciphertext E using K;
– verify the attribute-based signature for the access structure pid_0 and the tag tag_0; if the verification fails or if $\mathsf{pid}_0 \neq \mathsf{pid}_i$, the protocol is aborted.

Now, the recovered key K_0 is used for authentication as follows:

Modification of Round j, $j \neq 0$. If the protocol is not aborted, user U_i instead of broadcasting $m_{i,j}$ in protocol P, he will do the following:

– use K_0 to compute a tag $\mathsf{tag}_{i,j} := \mathtt{Tag}_{K_0}(m_{i,j}||\mathsf{nonces}_{U_i})$.
– broadcast $m_{i,j}||\mathsf{tag}_{i,j}$.

When receiving a message $m_{l,j}||\mathsf{tag}_{l,j}$, user U_i checks the following:

– $U_l \in \mathsf{pid}_0$
– j is the expected round number
– validity of the tag $\mathsf{tag}_{l,j}$.

If any of these checks fails, the protocol is aborted without accepting a session key. Otherwise, the session identifier is the concatenation of all messages sent and received by the protocol instance during its execution and the session key is as in P.

4.2 Security analysis

The following result affirms that given any protocol P the compiler described above adds authentication:

Proposition 1 *With the above notation, the attribute-based group key establishment obtained from the compiler in Section* **??** *is authenticated and secure in the sense of Definition* **??** *(in particular, forward secrecy is preserved).*

Proof We prove the security of the protocol by 'game hopping' [**?**], letting the probabilistic polynomial time adversary $\mathscr{A}$ of the compiled protocol P' interact with a simulator $\mathscr{S}$. The success probability respectively the advantage of $\mathscr{A}$ in Game i will be denoted by $\Pr[\mathsf{Succ}^{\text{Game } i}_{\mathscr{A}}]$ respectively $\mathrm{Adv}^{\text{Game } i}_{\mathscr{A}}$. By q_{send} we denote a polynomial upper bound for the number of queries by $\mathscr{A}$ to the Send oracle, and analogously we write q_{execute} for a polynomial upper bound on the number of queries to $\mathsf{Execute}$ made by $\mathscr{A}$.

Game 0: This game is identical to the original attack game, with all oracles of the adversary being simulated faithfully by $\mathscr{S}$. Consequently,

$$\mathrm{Adv}^{\text{ke}}_{\mathscr{A}} = \mathrm{Adv}^{\text{Game } 0}_{\mathscr{A}}.$$

Game 1: Let Repeat be the event that some user U_i uses a nonce r_i in Round 0 which this user has used previously already. This game is identical to Game 0 with the only exception that we abort the simulation and consider $\mathscr{A}$ as successful whenever the event Repeat occurs. Denoting by n a polynomial upper bound for the number of users given

as argument to a single Execute query, we have $\Pr[\mathsf{Repeat}] \leq (q_{\text{send}} + n \cdot q_{\text{execute}})^2/2^k$, and hence

$$|\mathrm{Adv}_{\mathscr{A}}^{\text{Game 1}} - \mathrm{Adv}_{\mathscr{A}}^{\text{Game 0}}| \leq \frac{(q_{\text{send}} + n \cdot q_{\text{execute}})^2}{2^k}$$

is negligible.

Game 2: In this game the simulator makes a uniform at random guess which instance will be queried to Test by $\mathscr{A}$ and also guesses which instance will initiate the corresponding protocol execution. Whenever such a guess turns out to be incorrect, we abort and consider the adversary $\mathscr{A}$ as successful. Otherwise this game is identical to the previous one. As $\mathscr{A}$ can involve at most $q_{\text{send}} + n \cdot q_{\text{execute}}$ instances, we have

$$\mathrm{Adv}_{\mathscr{A}}^{\text{Game 2}} \leq (q_{\text{send}} + n \cdot q_{\text{execute}})^2 \cdot \mathrm{Adv}_{\mathscr{A}}^{\text{Game 1}},$$

and it will suffice to recognize $\mathrm{Adv}_{\mathscr{A}}^{\text{Game 2}}$ as negligible.

Game 3: Let ForgeS be the event that, for the Test-instance, $\mathscr{A}$ succeeds in forging a new attribute-based signature for the initiator U_0 in Round 0 before querying $\mathsf{Corrupt}(U_i)$ for some $U_i \in \mathsf{pid}_0$. Game 3 is identical to Game 2 with the only exception that we abort the simulation and consider $\mathscr{A}$ as successful whenever the event ForgeS occurs. Occurrence of this event yields immediately an adversary $\mathscr{A}_{\text{uf-cmaa}}$ against the attribute-based signature scheme, and

$$|\mathrm{Adv}_{\mathscr{A}}^{\text{Game 3}} - \mathrm{Adv}_{\mathscr{A}}^{\text{Game 2}}| \leq \mathrm{Adv}_{\mathscr{A}_{\text{rsig}}}^{\text{uf-cmaa}}.$$

Game 4: This game is identical to Game 3 except that $\mathscr{S}$ produces the ciphertext E under an encryption of a freshly generated key $K' \leftarrow \mathtt{KeyGen}(1^k)$ instead of using the real key K. To bound $|\mathrm{Adv}_{\mathscr{A}}^{\text{Game 4}} - \mathrm{Adv}_{\mathscr{A}}^{\text{Game 3}}|$ we derive from $\mathscr{S}$ an algorithm $\mathscr{A}_{\text{abkem}}$ to attack the IND-AB-CCA security of the underlying EP-AB-KEM: $\mathscr{A}_{\text{abkem}}$ runs a simulation of $\mathscr{S}$ as in Game 3 and uses as $\mathbb{A}^*$ the Test-instance's partner identifier. The only modification in the simulation of $\mathscr{S}$ is for the initator U_0: denoting by $(\widetilde{K}_{b'}, C')$ the pair obtained from the EP-AB-KEM challenger, $\mathscr{A}_{\text{abkem}}$ replaces (K, C) with $(\widetilde{K}_b, C')$ in Round 0 and uses $\widetilde{K}_b$ to compute the ciphertext E.

Whenever $\mathscr{A}$ correctly identifies the session key after receiving the challenge of the (simulated) Test oracle, $\mathscr{A}_{\text{abkem}}$ outputs the guess $b' = 0$, whenever $\mathscr{A}$ guesses incorrectly, $\mathscr{A}_{\text{abkem}}$ outputs $b' = 1$.

Writing b^{ind} and b^{Test} for the values of the random bit used by the EP-AB-KEM challenger and the random bit of the (simulated) Test oracle, respectively, we get (with a slight abuse of notation) $\Pr[\mathsf{Succ}_{\mathscr{A}_{\text{abkem}}}^{\text{ind-ab-cca}}] =$

$$\begin{aligned}
&\frac{1}{2} \cdot \left(\frac{\Pr[1 \leftarrow \mathscr{A}^{b^{\mathsf{Test}}=1} \mid b^{\mathsf{ind}} = 0]}{2} + \frac{1 - \Pr[1 \leftarrow \mathscr{A}^{b^{\mathsf{Test}}=0} \mid b^{\mathsf{ind}} = 0]}{2} \right) \\
+ &\frac{1}{2} \cdot \left(\frac{1 - \Pr[1 \leftarrow \mathscr{A}^{b^{\mathsf{Test}}=1} \mid b^{\mathsf{ind}} = 1]}{2} + \frac{\Pr[1 \leftarrow \mathscr{A}^{b^{\mathsf{Test}}=0} \mid b^{\mathsf{ind}} = 1]}{2} \right) \\
= &\frac{1}{4} \cdot \left(\Pr[\mathsf{Succ}_{\mathscr{A}}^{\text{Game 4}}] - \Pr[\mathsf{Succ}_{\mathscr{A}}^{\text{Game 3}}] \right) + \frac{1}{2}
\end{aligned}$$

Consequently, $2 \cdot \mathrm{Adv}_{\mathscr{A}_{\text{abkem}}}^{\text{ind-ab-cca}} = 2 \cdot |2 \cdot \Pr[\mathsf{Succ}_{\mathscr{A}_{\text{abkem}}}^{\text{ind-ab-cca}}] - 1| =$

$$\left| \Pr[\mathsf{Succ}_{\mathscr{A}}^{\text{Game 4}}] - \Pr[\mathsf{Succ}_{\mathscr{A}}^{\text{Game 3}}] \right| \geq \left| \mathrm{Adv}_{\mathscr{A}}^{\text{Game 4}} - \mathrm{Adv}_{\mathscr{A}}^{\text{Game 3}} \right|.$$

Game 5: This game is identical to Game 4 except that in Round 0 of the protocol, $\mathscr{S}$ replaces the ciphertext E, sent by the initiator U_0 of the Test-instance, with an encryption of a uniformly at random chosen bitstring of the appropriate length. To bound $|\mathrm{Adv}_{\mathscr{A}}^{\text{Game 5}} - \mathrm{Adv}_{\mathscr{A}}^{\text{Game 4}}|$ we can derive from $\mathscr{A}$ an algorithm $\mathscr{A}_{\text{ror}}$ to attack the ROR-CCA security of the underlying symmetric encryption scheme: $\mathscr{A}_{\text{ror}}$ runs a simulation of Game 4 with the following exception: to create the ciphertext E for the initiator U_0 of the Test-instance, the 'composed oracle' $\mathscr{E}_K(\mathscr{R}\mathscr{R}(\cdot,b))$ is invoked, and for decrypting messages other than E under K, the decryption oracle $\mathscr{D}_K(\cdot)$ is invoked. The Corrupt, Execute, Reveal, Send and Test oracle for $\mathscr{A}$ can be simulated by $\mathscr{A}_{\text{ror}}$ in the obvious way.

Whenever $\mathscr{A}$ correctly identifies the session key after receiving the challenge of the (simulated) Test oracle, $\mathscr{A}_{\text{ror}}$ outputs 1, i. e., claims that its encryption oracle operates in 'real mode', whenever $\mathscr{A}$ guesses incorrectly, $\mathscr{A}_{\text{ror}}$ outputs 0. We obtain

$$|\mathrm{Adv}_{\mathscr{A}_{\text{ror}}}^{\text{ror-cca}}| = \left|\Pr[\mathsf{Succ}_{\mathscr{A}}^{\text{Game 4}}] - \Pr[\mathsf{Succ}_{\mathscr{A}}^{\text{Game 5}}]\right| \geq \left|\mathrm{Adv}_{\mathscr{A}}^{\text{Game 5}} - \mathrm{Adv}_{\mathscr{A}}^{\text{Game 4}}\right|.$$

Game 6: Let ForgeMAC denote the event that $\mathscr{A}$ succeeds in forging a new valid (message, tag)-pair for a user U_i before quering $\mathsf{Corrupt}(U_j)$ for some user $U_j \in \mathsf{pid}_j$. This game is identical to Game 5 with the only exception that we abort the simulation and consider $\mathscr{A}$ as successful whenever the event ForgeMAC occurs. Occurrence of this event yields immediately an adversary $\mathscr{A}_{\text{mac}}$ against the message authentication code, and

$$|\mathrm{Adv}_{\mathscr{A}}^{\text{Game 6}} - \mathrm{Adv}_{\mathscr{A}}^{\text{Game 5}}| \leq \mathrm{Adv}_{\mathscr{A}_{\text{mac}}}^{\text{suf-cma}}.$$

To conclude the proof, we show $\mathrm{Adv}_{\mathscr{A}}^{\text{Game 6}}$ is negligible by showing that the simulator $\mathscr{S}$ can, running a simulation of $\mathscr{A}$, be turned into a (passive) adversary against P. To attack P, first of all $\mathscr{S}$ corrupts each user in protocol P to obtain all long-term secrets. Moreover, $\mathscr{S}$ generates the master secret keys and the public parameters for the attribute-based signature scheme and for the EP-AB-KEM scheme and therewith has the long-term secrets of all parties in protocol P′. Making use of this information and maintaining a 'nonces and transcripts list' NT, the simulator $\mathscr{S}$ can run a simulation of Game 6 for $\mathscr{A}$ with oracle queries being answered as follows:

- **Execute queries.** When $\mathscr{A}$ makes a query Execute which does not involve the Test-instance, then $\mathscr{S}$ executes the protocol itself and stores the resulting nonces nonces_{U_i} along with a special symbol $\perp$ to indicate an empty transcript in the list NT. Knowing all long-term secrets, this simulation of Execute is perfect.
  If the Test instance is involved, $\mathscr{S}$ queries its own Execute oracle, and completes the resulting transcript T of protocol P in the obvious way to obtain a perfect simulation of Game 6, making use of the known long-term secrets of all parties. The respective nonces along with the 'padded' transcript T' are stored in the list NT.
- **Send queries.** We denote a query which initiates a new execution for an instance of a user U_i by Send_0. The second Send query to the same instance which includes the message of the form (U_j, r_j) for each user in the pid_i is denoted by Send_1. On a query $\mathsf{Send}_0(U_i, *)$, the simulator $\mathscr{S}$ chooses a random nonce $r_i \in \{0,1\}^k$ itself and replies with the respective Round 0 message. On a Send_1 query that is not directed to the Test instance, $\mathscr{S}$ computes nonces_{U_i}, stores $(\mathsf{nonces}_{U_i}, \perp)$ in the List NT, and answers by computing the next step of P′.
  If $\mathscr{S}$ receives a Send_1 query for the Test instance, then $\mathscr{S}$ first looks in its list NT for an entry of the form $(\mathsf{nonces}_{U_i}, T')$. If such an entry exists, then $\mathscr{S}$ takes the appropriate message from T' and answers to $\mathscr{A}$. If such an entry does not exist, then

$\mathscr{S}$ queries its Execute oracle, produces the transcript of P', stores T' in the list NT as it was done previously and answers $\mathscr{A}$ with the appropriate message.
- **Reveal queries.** From Game 2, $\mathscr{S}$ knows the Test instance, and hence can answer all queries to Reveal by computing the session key by itself.
- **Corrupt queries.** Since $\mathscr{S}$ knows all long-term secrets of users in the protocol P', $\mathscr{S}$ replies in the obvious way.
- **Test query.** From Game 2, $\mathscr{S}$ knows the Test instance, and can simply forward $\mathscr{A}$'s query to its own Test oracle.

In summary, the simulator $\mathscr{S}$ answers all queries of $\mathscr{A}$ exactly as in Game 6, and whenever $\mathscr{A}$ violates the security of P', then $\mathscr{S}$ violates the security of P:

$$\mathrm{Adv}_{\mathscr{A}}^{\text{Game 6}} \leq \mathrm{Adv}_{\mathscr{S}}^{\mathrm{ke}}.$$

□

The compiler in Section **??** can be applied to any protocol P. If P is fully authenticated, where parties sign all messages, we cannot expect that the compiled protocol is deniable, as the simulator $\mathscr{S}_d$ in the deniability definition does not have access to signing keys of uncorrupted users. We aim at applying our compiler to protocols that are 'only' passively secure and add authentication whereas remaining deniable. The more typical passively secure protocol does not involve any long-term secrets, and in these cases the following result ensures the deniability of P'.

Proposition 2 *If the attribute-based group key establishment* P *does not involve long-term secrets, then the attribute-based group key establishment* P' *obtained by applying the compiler in Section* **??** *to* P *is deniable in the sense of Definition* **??**.

Proof To prove the deniability of the protocol, we use the usual 'game hopping' technique [**?**], letting the probabilistic polynomial time adversary $\mathscr{A}_d$ of the compiled protocol P' and the simulator $\mathscr{S}_d$ interact with the challenger $\mathscr{C}$. The advantage of the distinguisher $\mathscr{X}$ in Game i will be denoted by $\mathrm{Adv}_{\mathscr{X}}^{\text{Game } i}$.

Game 0: This game is identical to the original deniability game, with all oracles of the adversary and simulator being simulated faithfully by $\mathscr{C}$. Consequently,

$$\mathrm{Adv}_{\mathscr{X}}^{\mathrm{den}} = \mathrm{Adv}_{\mathscr{X}}^{\text{Game 0}}.$$

Game 1: This game is identical to Game 0 except that in the simulation of the Send oracle, $\mathscr{C}$ changes Round 0 messages for the initiator U_0 if no participant $U_i \in \mathsf{pid}_0$ has been corrupted (i. e., the adversary does not have any private key (sk_i, dk_i)). In this case, to compute the ciphertext E, the simulator encrypts just a random bitstring of the appropriate length.

To argue that the advantage of the distinguisher $\mathscr{X}$ in Game 0 and Game 1 differs only negligibly, consider the following adversary $\mathscr{D}$ against the real-or-random indistinguishability of the symmetric encryption scheme: the algorithm $\mathscr{D}$ will take the role of $\mathscr{C}$ and act as challenger for $\mathscr{A}_d$ and $\mathscr{S}_d$. In addition, $\mathscr{D}$ runs $\mathscr{X}$ as a subroutine. To initialize $\mathscr{A}_d$ and $\mathscr{S}_d$, the adversary $\mathscr{D}$ uses parameters he creates (honestly) on his own. Further, $\mathscr{D}$ simulates all of $\mathscr{A}_d$'s and $\mathscr{S}_d$'s protocol instances faithfully, except when answering queries of the form $\mathsf{Send}(U_0, s_0, \mathsf{pid}_0)$ from $\mathscr{A}_d$ to initiate the protocol with a partner identifier that involves no corrupted users. To answer the latter, he uses the real-or-random oracle instead of the encryption algorithm for determining

$E = \mathtt{Enc}_K(K_0||\mathsf{pid}_0||U_0||0||r_0||\sigma)$. He will use the output of this oracle to answer the query. Finally, $\mathscr{D}$ chooses $b \in \{0,1\}$ uniformly at random.
If $\mathscr{X}$ outputs a correct guess $b' = b$, then $\mathscr{D}$ outputs 1, otherwise $\mathscr{D}$ outputs 0. Therefore,

$$\left|\mathrm{Adv}^{\text{Game 0}}_{\mathscr{X}} - \mathrm{Adv}^{\text{Game 1}}_{\mathscr{X}}\right| \leq |\mathrm{Adv}^{\text{ror-cca}}_{\mathscr{D}}(k)|,$$

which is negligible.

Game 2: This game is identical to Game 1 except that $\mathscr{C}$ changes the simulation of the Send oracle for Round 0 messages for the initiator U_0 if some participant $U_j \in \mathsf{pid}_0$ has been corrupted. In this case, the adversary has a secret key pair (sk_j, dk_j) (if he has more than one secret key pair, he selects one at random). The challenger faithfully simulates all computations of U_0 using sk_j to compute the required attribute-based signature. If the distinguisher $\mathscr{X}$ notices the difference between this simulation and the one in Game 2, he could be used as blackbox to attack the privacy of the attribute-based signature. More specifically, let $\mathscr{F}$ be the following adversary against the privacy of the attribute-based signature:
The algorithm $\mathscr{F}$ will act as challenger for $\mathscr{A}_d$ and $\mathscr{S}_d$, and also run a simulation of $\mathscr{X}$. Further, $\mathscr{F}$ is given the master secret key, and initiates $\mathscr{A}_d$ and $\mathscr{S}_d$ with the corresponding keys—he creates the other parameters on his own. The algorithm $\mathscr{F}$ chooses an instance $\prod^s_U$ and an initiator $U_{i_0} \in \mathsf{pid}^s_U$ at random and simulates all of $\mathscr{A}_d$'s and $\mathscr{S}_d$'s instances faithfully, except when answering the query $\mathsf{Send}(\mathsf{pid}^s_U, U_{i_0})$ to initiate the protocol in the selected instance. In order to answer this query, $\mathscr{F}$ runs MKeyGen to get a key K_0. He will hand its challenger the message $m^* = K_0||\mathsf{pid}^s_U||U_{i_0}||0||r_0$, the access structure pid^s_U and two indices i_0 (the initiator) and i_1 such that $U_{i_0}, U_{i_1} \in \mathsf{pid}^s_U$. Hereafter, $\mathscr{F}$ will receive a challenge σ^*, and $\mathscr{F}$ will simulate all instances faithfully, except when the answer includes $\mathtt{SSign}_{sk_0}(K_0||\mathsf{pid}^s_U||U_{i_0}||0||r_0)$. In this case, he will substitute this value by σ^*. Notice $\mathscr{F}$ can query Corrupt and SSign for different messages, so he can simulate all other messages and Corrupt queries faithfully. Finally, $\mathscr{F}$ chooses $b \in \{0,1\}$ uniformly at random. If $\mathscr{X}$ outputs a correct guess $b' = b$, then $\mathscr{F}$ outputs 0, otherwise $\mathscr{F}$ outputs 1. Therefore,

$$\left|\mathrm{Adv}^{\text{Game 1}}_{\mathscr{X}} - \mathrm{Adv}^{\text{Game 2}}_{\mathscr{X}}\right| \leq 2 \cdot \mathrm{Adv}^{\text{absig-pri}}_{\mathscr{F}}(k),$$

which is negligible.
Notice that the simulation provided now to $\mathscr{A}_d$ by the challenger $\mathscr{C}$ is the same $\mathscr{S}_d$ would provide, thus the distinguisher's advantage in this case is 0.
Collecting all the advantages together, we get:

$$\mathrm{Adv}^{\text{den}}_{\mathscr{X}} \leq |\mathrm{Adv}^{\text{ror-cca}}_{\mathscr{D}}(k)| + 2 \cdot \mathrm{Adv}^{\text{absig-pri}}_{\mathscr{F}}(k),$$

which is negligible.

□

4.3 Example: a deniable 3-round protocol with forward secrecy

The most natural scenario for applying our compiler is to start out with an unauthenticated group key establishment where no public key infrastructure is involved. A possible candidate is the Burmester-Desmedt protocol [**?**] which has already been used by Katz and Yung [**?**] to derive an authenticated key establishment in the ordinary public-key setting, not taking

into account deniability. We notice that the Burmester-Desmedt protocol provides forward secrecy in our model—our partnering definition prevents the adversary from attacking more instances than in [**?**]. So applying our compiler yields a 3-round solution for authenticated attribute-based group key establishment, and this 3-round protocol provides forward secrecy as well as deniability.

Better round efficiency can be achieved if we are willing to sacrifice some of the properties: a one-round protocol [**?**] is available if forward secrecy is not required. The two-round proposal [**?**] falls short from the point of view of deniability: there, the protocol initiator sends a signcrypted message. So even if the underlying signcryption scheme provides some form of privacy, it is not clear how a simulator should simulate the output of an Execute query when no user is corrupted.

5 Conclusion

In this paper we proposed a *general compiler* to derive authenticated attribute-based group key establishment protocols from unauthenticated ones. Forward secrecy is preserved, and the constructed protocols offer deniability, as long as the protocol we start with does not involve long-term secrets. In terms of round complexity, one additional round is added, just as in the Katz-Yung compiler. In particular starting out with a passively secure 2-round solution, we obtain the first example of a 3-round solution for attribute-based key establishment offering both forward secrecy and deniability.

The technique we used seems also to be of interest for other forms of authentication. E. g., when looking at a predicate-based scenario (cf. [**?**]) it appears plausible to adopt our approach, provided that suitable signature and key encapsulation primitives are available.

Acknowledgements The authors would like to thank an anonymous reviewer for constructive feedback that helped to improve the original manuscript.