Enhanced Security of Attribute-Based Signatures

Johannes Blömer Fabian Eidens Jakob Juhnke

September 28, 2018
Department of Computer Science
Paderborn University, Germany
{bloemer,feidens, juhnke}@mail.uni-paderborn.de

Abstract

Despite the recent advances in attribute-based signatures (ABS), no schemes have yet been considered under a strong privacy definition. We enhance the security of ABS by presenting a strengthened simulation-based privacy definition and the first attribute-based signature functionality in the framework of universal composability (UC). Additionally, we show that the UC definition is equivalent to our strengthened experiment-based security definitions. To achieve this we rely on a general unforgeability and a simulation-based privacy definition that is stronger than standard indistinguishability-based privacy. Further, we show that two extant concrete ABS constructions satisfy this simulation-based privacy definition and are therefore UC secure. The two concrete constructions are the schemes by Sakai et al. (PKC'16) and by Maji et al. (CT-RSA'11). Additionally, we identify the common feature that allows these schemes to meet our privacy definition, giving us further insights into the security requirements of ABS.

Keywords: Attribute-Based Signatures, Privacy, Universal Composability

The authors were partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB 901).

The authors were partially supported by the Ministry of Education and Research, grant 16SV7055, project "KogniHome".

Contents

1 Introduction

Attribute-based signature schemes and an experiment-based security definition were introduced by Maji, Prabhakaran and Rosulek [MPR11]. The concept of attribute-based signatures considers several signers and an authority that issues secret keys to them. Secret keys encode an attribute set. Attribute-based signatures are computed on message-policy pairs under a secret key. Policies are for example Boolean formulas over the attributes. To generate a valid signature a signer has to possess a secret key where the encoded attributes satisfy the given policy. Given an attribute-based signature everyone is able to verify whether it was generated by a signer that possesses attributes satisfying the given policy. The validity of a signature is therefore not bound to a single signer's identity but rather to a group of signers, namely those with satisfying attributes.

A secure ABS must be unforgeable and (perfectly) private. Unforgeability means that a valid signature on a message-policy pair can only be generated by a signer, whose attributes satisfy the policy. Further, no group of colluding signers can generate a signature on a message-policy pair if none of them has attributes satisfying the policy. Privacy captures that a signature is independent of the secret key used to generate it.

1.1 Related Work

Throughout the literature experiment-based security definitions covering unforgeability and privacy for ABS schemes have been proposed, cf. [BF14; EHM11; MPR11; OT14; SAH16]. A general unforgeability definition is given by Okamoto and Takashima [OT14], where the definitions in [EHM11; MPR11; SAH16] are restricted to specific policy classes.

Regarding privacy there are two definitions, perfect and computational privacy. Computational privacy is defined by an experiment with explicit capabilities of bounded adversaries, cf. [BF14; EHM11; Her16]. For unbounded adversaries perfect privacy is defined considering distributions of signatures [MPR08; MPR11; OT14; SAH16]. In particular one demands that the distributions are independent of the secret keys. Another notion of privacy is called simulation-based privacy and was originally presented by Bellare and Fuchsbauer [BF14] in the context of policy-based signatures, a more general concept than ABS. The authors [BF14] also show that policy-based signatures imply attribute-based signatures in the model of Maji et al. [MPR11]. Hence their security definitions can be specialized to ABS.

Recent advances in ABS led to efficient schemes that support large classes of policies. The recent scheme by Sakai et al. [SAH16] supports arbitrary circuits of unbounded size and depth. A further efficient scheme is presented by Okamoto and Takashima [OT14], supporting non-monotone span programs as policies. A generic ABS construction with monotone span programs as policies is presented by Maji et al. [MPR11]. All these ABS schemes and security definitions look at ABS as an isolated primitive. However, in real applications ABS is combined and composed with other cryptographic primitives to achieve more comprehensive security goals. For example, ABS can be deployed as an authentication mechanism for service providers, i.e. in a challenge-response protocol. In such a protocol the user is asked to sign a policy and a nonce given by the service provider. Such authentication mechanisms are deployed in large scale applications. Canetti [Can01] introduced the universal composable framework (UC) to describe

and prove security for such applications. UC guarantees security even if an arbitrary number of cryptographic primitives are executed concurrently and it is based on the simulation-paradigm.

Regarding the experiment-based and UC security of a cryptographic primitive there are four questions that have to be answered. First, is there a general experiment-based security definition that precisely captures the security aspect of the cryptographic primitive? Second, whether there exist schemes that are secure with respect to the experiment-based security definition. The third question is, how to define security for that primitive in the UC framework? The last question is, whether the experiment-based security definition and the UC security definition are equivalent. With respect to ABS the first question can be answered by the general unforgeability definition in [OT14] and the simulation-based privacy definition presented in [BF14]. However, the answer is not yet satisfactory for achieving UC security. To the best of our knowledge, no one has yet answered the remaining three questions in the context of ABS. However the three questions have been considered for other related primitives. With respect to the third question Ateniese et al. [Ate+05] construct a UC secure group signature scheme and Camenisch et al. [Cam+15] present a UC secure anonymous credential system. Regarding the fourth question Canetti shows in [Can03] that his ideal digital signature functionality is equivalent to the standard security definition (EUF-CMA) [GMR88].

1.2 Our Contribution

In this paper we answer the four questions for attribute-based signatures. First, we give a strengthened experiment-based security definition using a simulation-based privacy definition. Second, we show that existing schemes satisfy this definition. Third, we model the first (to our knowledge) universally composable attribute-based signature functionality. Considering the fourth question, our main theorem shows the equivalence of the experiment-based and UC security of attribute-based signatures under standard requirements on the environment. The theorem is shown considering adaptive corruption of parties with erasure of ephemeral randomness and secure communication channels. We explain the requirements in detail in Section 3.1. Our results show that existing ABS schemes achieve UC security with only minor modifications.

In this paper we argue that the security definitions for ABS used so far do not guarantee the desired privacy and as a consequence our definition should be used. Our experiment-based security definition is based on existing unforgeability and privacy definitions. It incorporates the general ABS unforgeability definition by Okamoto and Takashima [OT14]. For our perfect simulation-based privacy definition we use as a basis the computational simulation privacy definition by Bellare and Fuchsbauer [BF14]. The authors [BF14] show for policy-based signatures that *computational* simulation privacy is stronger than *computational* privacy used in [AAS16; EHM11; MPR08; MPR11; OT14; SAH16]. Based on our security definitions for ABS we show that *perfect* simulation privacy is stronger than *perfect* privacy.

Our UC ideal ABS functionality is based on the ideal digital signatures from Canetti [Can03], extended to support multiple signing parties and signature creation on message-policy pairs under secret keys with attributes. Therefore, our ideal ABS functionality is one of a few functionalities that consider multiple parties concurrently executing cryptographic tasks. Other such functionalities include the multi-commitments by Lindell [Lin11], the group signatures by

Ateniese et al. [Ate+05], and the anonymous credentials by Camenisch et al. [Cam+15].

Further, we show that our experiment-based security definition implies UC security for attribute-based signature schemes with minimal restrictions on the environment. In our proof the simulation aspect of the privacy definition is the essential ingredient. We also show the reverse direction, i.e. a UC secure ABS scheme also satisfies our experiment-based security definition. The proof of this result is inspired by the work of Abe and Ohkubo [AO12] on UC secure blind signature schemes. To show the applicability of our results we prove that the generic ABS constructions by Sakai et al. [SAH16] and by Maji et al. [MPR11] satisfy our simulation-based privacy definition and therefore achieve UC security. Since originally for both schemes weaker privacy guarantees were shown, we think that simulation-based privacy is interesting on its own.

2 Attribute-Based Signatures

In the following we formally define attribute-based signature (ABS) schemes and their experiment-based security notions.

Definition 2.1. An ABS scheme Π_{ABS} consists of four algorithms:

Setup(1^λ) Probabilistic algorithm that takes as input security parameter 1^λ and outputs public parameters and master secret $(\mathsf{pp}, \mathsf{msk})$, where pp includes a description of the attribute universe denoted by $\mathbb{U}(\mathsf{pp})$.

KeyGen$(\mathsf{pp}, \mathsf{msk}, \mathbb{A})$ Probabilistic algorithm that takes as input pp, msk, and a set of attributes $\mathbb{A} \subseteq \mathbb{U}(\mathsf{pp})$. It outputs a signing key $\mathsf{sk}_{\mathbb{A}}$.

Sign$(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}}, m, \mathbb{P})$ Probabilistic algorithm that takes as input public parameters pp, a message m, a policy $\mathbb{P}$ over the attributes of $\mathbb{U}(\mathsf{pp})$, and a signing key $\mathsf{sk}_{\mathbb{A}}$, such that $\mathbb{P}(\mathbb{A}) = 1$. It outputs a signature σ.

Verify$(\mathsf{pp}, m, \mathbb{P}, \sigma)$ Probabilistic algorithm that takes as input pp, a message m, a policy $\mathbb{P}$, and a signature σ. It outputs $b \in \{0, 1\}$; valid $= 1$, invalid $= 0$.

How the universe, attributes, and policies are encoded and interpreted depends on the concrete scheme. We denote attributes $\mathbb{A}$ as subsets of $\mathbb{U}(\mathsf{pp})$ and a policy $\mathbb{P}$ is a map $\mathbb{P} : 2^{\mathbb{U}(\mathsf{pp})} \to \{0, 1\}$. Alternatively, we could also write $\mathbb{A} \in \mathbb{U}(\mathsf{pp})$ and $\mathbb{P} : \mathbb{U}(\mathsf{pp}) \to \{0, 1\}$ which is better suited for ABS schemes supporting attribute vectors of fixed length as in [SAH16]. With $\mathbb{P} \in \mathbb{U}(\mathsf{pp})$ we denote that the policy $\mathbb{P}$ is defined over the attributes of the universe $\mathbb{U}(\mathsf{pp})$.

Definition 2.2 (Correctness)**.** An scheme Π_{ABS} is correct, if for all $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathrm{Setup}(1^\lambda)$, all m, all attribute sets $\mathbb{A} \subseteq \mathbb{U}(\mathsf{pp})$, all $\mathsf{sk}_{\mathbb{A}} \leftarrow \mathrm{KeyGen}(\mathsf{pp}, \mathsf{msk}, \mathbb{A})$, all policies $\mathbb{P} \in \mathbb{U}(\mathsf{pp})$ such that $\mathbb{P}(\mathbb{A}) = 1$, and all $\sigma \leftarrow \mathrm{Sign}(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}}, m, \mathbb{P})$, it holds that $\mathrm{Verify}(\mathsf{pp}, m, \mathbb{P}, \sigma) = 0$ with at most negligible probability $\epsilon(\lambda)$ where the probability is over the random choices of Sign and Verify.

Correctness guarantees under honestly generated setup parameters, that signatures, computed with honestly generated secret keys, on message-policy pairs are valid with overwhelming probability.

Definition 2.3 (Consistency)**.** An ABS scheme Π_{ABS} is consistent, if for all m, $\mathbb{P}$, and σ it holds that there exists $b \in \{0,1\}$ such that $\Pr[(\mathsf{pp},\mathsf{msk}) \leftarrow \mathrm{Setup}(1^\lambda)\colon \mathrm{Verify}(\mathsf{pp}, m, \mathbb{P}, \sigma) \neq b] \leq \epsilon(\lambda)$, where $\epsilon(\lambda)$ is negligible in λ and the probability is over randomness of Setup and Verify.

Our definition of ABS considers a probabilistic Verify algorithm. Therefore, consistency guarantees that a signature that was once declared by Verify as valid will be declared as invalid by an independent run of Verify only with negligible probability (and vice versa).

Definition 2.4. For a forger F, we define the experiment $\mathrm{Exp}_F^{\mathrm{UF}}(\lambda)$ as follows:

1. Run $(\mathsf{pp},\mathsf{msk}) \leftarrow \mathrm{Setup}(1^\lambda)$, start $F(\mathsf{pp})$.
2. F may adaptively make queries of the following type:
   $\mathcal{O}^{\mathrm{KeyGen}}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_i)$: On i-th query, given attribute set $\mathbb{A}_i \subseteq \mathbb{U}(\mathsf{pp})$ the oracle generates $\mathsf{sk}_{\mathbb{A}_i} \leftarrow \mathrm{KeyGen}(\mathsf{pp},\mathsf{msk},\mathbb{A}_i)$ and records $(i, \mathsf{sk}_{\mathbb{A}_i})$.

   $\mathcal{O}^{\mathrm{Reveal}}(i)$: Given i specifying an already queried secret key for $\mathbb{A}_i$, it outputs the corresponding secret key $\mathsf{sk}_{\mathbb{A}_i}$.

   $\mathcal{O}^{\mathrm{Sign}}_{\mathsf{pp},\mathsf{msk}}(i, m_j, \mathbb{P}_j)$: On j-th query, given $(i, m_j, \mathbb{P}_j)$ for an already queried $\mathbb{A}_i$ where $\mathbb{P}_j(\mathbb{A}_i) = 1$, $\mathbb{P}_j \in \mathbb{U}(\mathsf{pp})$, it returns $\sigma \leftarrow \mathrm{Sign}(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$ for message m_j, policy $\mathbb{P}_j$ and secret key $\mathsf{sk}_{\mathbb{A}_i}$.
3. Eventually F outputs a triple $(m^*, \mathbb{P}^*, \sigma^*)$
4. The output is 1, if all of the following conditions hold, else it is 0,
   a) a signature for $(m^*, \mathbb{P}^*)$ was never queried and
   b) for all $\mathbb{A}_i$, where the corresponding secret key $\mathsf{sk}_{\mathbb{A}_i}$ was output by the reveal oracle $\mathcal{O}^{\mathrm{Reveal}}$, it holds that $\mathbb{P}^*(\mathbb{A}_i) \neq 1$, and
   c) $\mathrm{Verify}(\mathsf{pp}, m^*, \mathbb{P}^*, \sigma^*) = 1$.

Definition 2.5 (Unforgeability)**.** An ABS scheme is unforgeable regarding an adaptive attack if for all ppt forger F, $\Pr\left[\mathrm{Exp}_F^{\mathrm{UF}}(\lambda) = 1\right]$ is negligible in λ.

The above definition originally presented in [OT14] guarantees collusion resistance in the following sense. The adversary can get secret keys on attribute sets of his choice by first querying $\mathcal{O}^{\mathrm{KeyGen}}_{\mathsf{pp},\mathsf{msk}}$ and then $\mathcal{O}^{\mathrm{Reveal}}$. Even with secret keys of his choice he can not output a valid signature for $\mathbb{P}^*$ if none of the revealed secret keys (representing a group of colluding signers) would be sufficient to satisfy $\mathbb{P}^*$.

2.1 Privacy

We present two privacy definitions. The first one captures that an adversary choosing two secret keys should not be able to tell which secret key was used to generate a signature. The second definition is simulation-based and requires that even the attributes used to generate a signature are hidden. For both definitions we specialize the definitions for policy-based signatures presented in [BF14] to ABS.

2.1.1 Standard Privacy

Contrary to the privacy definition based on distributions of signatures used in [Her16; MPR11; OT14; SAH16] we define an experiment similar to [BF14], where the capabilities of the adversary are explicitly stated.

Definition 2.6. For an ABS scheme $\Pi_{\mathrm{ABS}} = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ and dist. D we define the privacy experiment $\mathrm{P}_D^{\Pi_{\mathrm{ABS}}}(\lambda, b)$ for $b \in \{0,1\}$ as follows.

$\mathrm{P}_D^{\Pi_{\mathrm{ABS}}}(\lambda, b)$:
$(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathrm{Setup}(1^\lambda)$
$\tilde{b} \leftarrow D^{\mathcal{O}^b_{\mathsf{pp},\mathsf{msk}}}(\mathsf{pp}, \mathsf{msk})$
return $\tilde{b}$

$\mathcal{O}^b_{\mathsf{pp},\mathsf{msk}}(m, \mathbb{P}, \mathbb{A}_0, \mathbb{A}_1)$:
If $\mathbb{P}(\mathbb{A}_0) = 1$, $\mathbb{P}(\mathbb{A}_1) = 1$, and $\mathbb{P} \in \mathbb{U}(\mathsf{pp})$, it generates:
$\mathsf{sk}_{\mathbb{A}_0} \leftarrow \mathrm{KeyGen}(\mathsf{pp}, \mathsf{msk}, \mathbb{A}_0)$,
$\mathsf{sk}_{\mathbb{A}_1} \leftarrow \mathrm{KeyGen}(\mathsf{pp}, \mathsf{msk}, \mathbb{A}_1)$,
$\sigma \leftarrow \mathrm{Sign}(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}_b}, m, \mathbb{P})$,
and returns $(\sigma, \mathsf{sk}_{\mathbb{A}_0}, \mathsf{sk}_{\mathbb{A}_1})$

Definition 2.7 (Privacy)**.** For an ABS scheme $\Pi_{\mathrm{ABS}} = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ we define:

Perfect Privacy Π_{ABS} is PP if for every distinguisher D it holds that

$$\mathrm{Adv}^{\mathrm{PP}}_D(\lambda) = \left|\Pr\left[\mathrm{P}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 0) = 1\right] - \Pr\left[\mathrm{P}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 1) = 1\right]\right| = 0 \ .$$

Computational Privacy Π_{ABS} is CP, if for every ppt dist. D it holds that

$$\mathrm{Adv}^{\mathrm{CP}}_D(\lambda) = \left|\Pr\left[\mathrm{P}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 0) = 1\right] - \Pr\left[\mathrm{P}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 1) = 1\right]\right| = \epsilon(\lambda) \ ,$$

where $\epsilon(\lambda)$ is a negligible function in λ.

The privacy in Definition 2.7 only states that the relation between the signature and the secret keys is hidden. In particular, an adversary can not determine which signer issued a signature. Another way to describe privacy, is that given a valid signature σ on a policy $\mathbb{P}$ an adversary should not be able to learn which attributes are necessary to satisfy $\mathbb{P}$ from σ, except for what it can compute from $\mathbb{P}$. Simulation privacy achieves this privacy level. To argue why simulation privacy is desirable in practice consider the following example. A similar example is presented in [BF14] for policy-based signatures. For the example assume a perfectly private ABS scheme according to Definition 2.7. Additionally, assume for every $\mathbb{P}$ there is just one satisfying attribute set $\mathbb{A}$. Hence, given a policy $\mathbb{P}$ the adversary knows the corresponding satisfying $\mathbb{A}$. Therefore, the adversary in experiment $\mathrm{P}^{\Pi_{\mathrm{ABS}}}$ has to input $\mathbb{A}_0 = \mathbb{A}_1$ for $\mathbb{P}$ to its challenge oracle $\mathcal{O}^b_{\mathsf{pp},\mathsf{msk}}$. Let us modify algorithm Sign such that it appends the attribute set to each signature. As a result, the returned signatures are still indistinguishable as required in Definition 2.7, but the used attribute set is known after a signature is shown. This is not the desired privacy guarantee in a real world application, where the attributes are secret or the satisfying set of a policy is a secret or hard to compute. To achieve the privacy level that is demanded in such applications we define simulation privacy.

2.1.2 Simulation Privacy

With the simulation-based definition of privacy we require that the signatures are independent of the used attributes. Therefore, simulation privacy is based on a simulation signature algorithm. The normal signature algorithm in ABS gets a secret key for an attribute set as an input, whereas the simulation signature algorithm does not. No adversary should be able to distinguish whether a signature was generated by the normal signature algorithm or by the simulation signature algorithm. Obviously, if signatures can be simulated without a given secret key for satisfying attributes (regarding the given policy), then the signatures themselves does not leak any information about the attributes used to generate it. The following simulation-based definition is originally presented in [BF14] for policy-based signatures.

Definition 2.8. For an ABS scheme $\Pi_{\mathrm{ABS}} = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$, a 3-tuple of ppt algorithms $(\mathrm{SimSetup}, \mathrm{SimKeyGen}, \mathrm{SimSign})$ and distinguisher D we define the simulation privacy experiment $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, b)$ for $b \in \{0,1\}$ as follows.

$\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 1)$:	$\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 0)$:
$(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathrm{Setup}(1^\lambda)$	$(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathrm{SimSetup}(1^\lambda)$
$\tilde{b} \leftarrow D^{\mathcal{O}^{\mathrm{KeyGen}_1}_{\mathsf{pp},\mathsf{msk}}, \mathcal{O}^{\mathrm{Sign}_1}_{\mathsf{pp},\mathsf{msk}}}(\mathsf{pp}, \mathsf{msk})$	$\tilde{b} \leftarrow D^{\mathcal{O}^{\mathrm{KeyGen}_0}_{\mathsf{pp},\mathsf{msk}}, \mathcal{O}^{\mathrm{Sign}_0}_{\mathsf{pp},\mathsf{msk}}}(\mathsf{pp}, \mathsf{msk})$
return $\tilde{b}$	return $\tilde{b}$

$\mathcal{O}^{\mathrm{KeyGen}_1}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_i)$: On i-th query, given attribute set $\mathbb{A}_i \subseteq \mathbb{U}(\mathsf{pp})$ it outputs $\mathsf{sk}_{\mathbb{A}_i} \leftarrow \mathrm{KeyGen}(\mathsf{pp}, \mathsf{msk}, \mathbb{A}_i)$ and records $(i, \mathsf{sk}_{\mathbb{A}_i})$.

$\mathcal{O}^{\mathrm{KeyGen}_0}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_i)$: On i-th query, given attribute set $\mathbb{A}_i \subseteq \mathbb{U}(\mathsf{pp})$ it outputs $\mathsf{sk}_{\mathbb{A}_i} \leftarrow \mathrm{SimKeyGen}(\mathsf{pp}, \mathsf{msk}, \mathbb{A}_i)$ and records $(i, \mathsf{sk}_{\mathbb{A}_i})$.

$\mathcal{O}^{\mathrm{Sign}_1}_{\mathsf{pp},\mathsf{msk}}(i, m_j, \mathbb{P}_j)$: On j-th query, given $(i, m_j, \mathbb{P}_j)$ for an already recorded i where $\mathbb{P}_j(\mathbb{A}_i) = 1$, and $\mathbb{P}_j \in \mathbb{U}(\mathsf{pp})$ it returns $\sigma \leftarrow \mathrm{Sign}\left(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}_i}, m_j, \mathbb{P}_j\right)$.

$\mathcal{O}^{\mathrm{Sign}_0}_{\mathsf{pp},\mathsf{msk}}(i, m_j, \mathbb{P}_j)$: On j-th query, given $(i, m_j, \mathbb{P}_j)$ for an already recorded i where $\mathbb{P}_j(\mathbb{A}_i) = 1$, and $\mathbb{P}_j \in \mathbb{U}(\mathsf{pp})$ it ignores $\mathbb{A}_i$ and $\mathsf{sk}_{\mathbb{A}_i}$. It returns signature $\sigma \leftarrow \mathrm{SimSign}\left(\mathsf{pp}, \mathsf{msk}, m_j, \mathbb{P}_j\right)$.

Definition 2.9 (Simulation Privacy)**.** For an attribute-based signature scheme $\Pi_{\mathrm{ABS}} = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ we define:

Perfect Simulation Privacy Π_{ABS} is PSimP if there exists a 3-tuple of ppt algorithms called $(\mathrm{SimSetup}, \mathrm{SimKeyGen}, \mathrm{SimSign})$ such that for all distinguisher D it holds that

$$\mathrm{Adv}_D^{\mathrm{PSimP}}(\lambda) = \left|\Pr\left[\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 0) = 1\right] - \Pr\left[\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 1) = 1\right]\right| = 0 \ .$$

Comp. Simulation Privacy Π_{ABS} is CSimP if there exists a 3-tuple of ppt algorithms called $(\mathrm{SimSetup}, \mathrm{SimKeyGen}, \mathrm{SimSign})$ such that for all ppt distinguisher D it holds that

$$\mathrm{Adv}_D^{\mathrm{CSimP}}(\lambda) = \left|\Pr\left[\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 0) = 1\right] - \Pr\left[\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 1) = 1\right]\right| = \epsilon(\lambda) \ ,$$

where $\epsilon(\lambda)$ is a negligible function in λ.

Theorem 2.1. *An ABS scheme Π_{ABS} that is perfectly simulation private (Definition 2.9) is also perfectly private (Definition 2.7).*

Proof. Assume that Π_{ABS} is not perfectly private then there is a dist. A such that the advantage $\mathrm{Adv}_A^{\mathrm{PP}}(\lambda) = \Pr[\mathrm{P}_A^{\Pi_{\mathrm{ABS}}}(\lambda,1)=1] - \Pr[\mathrm{P}_A^{\Pi_{\mathrm{ABS}}}(\lambda,0)=1] > 0$. We construct a distinguisher D for PSimP using A as a black-box. D in $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda,b)$ works as follows.

1. Given $(\mathsf{pp},\mathsf{msk})$ from challenger C of $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda,b)$, D runs $A(\mathsf{pp},\mathsf{msk})$.
2. D flips a coin $d \leftarrow \{0,1\}$.
3. D on the k-th oracle query $(m,\mathbb{P},\mathbb{A}_0,\mathbb{A}_1)$ where $\mathbb{A}_0,\mathbb{A}_1 \subseteq \mathbb{U}(\mathsf{pp})$:
   a) It checks whether $\mathbb{P}_i(\mathbb{A}_0^*)=1$ and $\mathbb{P}_i(\mathbb{A}_1^*)=1$ hold, if not ignore query
   b) Get $\mathsf{sk}_{\mathbb{A}_0} \leftarrow \mathcal{O}^{\mathrm{KeyGen}_b}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_0)$ and $\mathsf{sk}_{\mathbb{A}_1} \leftarrow \mathcal{O}^{\mathrm{KeyGen}_b}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_1)$
   c) Get $\sigma_d \leftarrow \mathcal{O}^{\mathrm{Sign}_b}_{\mathsf{pp},\mathsf{msk}}(2k-1+d,m,\mathbb{P})$
   d) Return $(\sigma_d,\mathsf{sk}_{\mathbb{A}_0},\mathsf{sk}_{\mathbb{A}_1})$
4. Eventually A outputs $\tilde{d}$.
5. D sets $\tilde{b} \coloneqq 1$ if $d=\tilde{d}$, otherwise $\tilde{b} \coloneqq 0$ and outputs $\tilde{b}$.

Let us analyze the advantage of D, $\mathrm{Adv}_D^{\mathrm{PSimP}}(\lambda) = |\Pr[\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda,0)=1] - \Pr[\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda,1)=1]|$. Let us first analyze the case where D is in the experiment $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda,b=1)$. With $b=1$ the challenger and the provided oracles use the normal algorithms $(\mathrm{Setup},\mathrm{KeyGen},\mathrm{Sign})$. Hence, we get the following.

$$\begin{aligned}
\Pr\left[\tilde{b}=1\right] &= \Pr\left[\tilde{d}=1 \mid d=1\right]\cdot\Pr[d=1] + \Pr\left[\tilde{d}=0 \mid d=0\right]\cdot\Pr[d=0] \\
&= \frac{1}{2}\left(\Pr\left[\tilde{d}=1 \mid d=1\right] - \Pr\left[\tilde{d}=1 \mid d=0\right]\right) + \frac{1}{2} \\
&= \frac{1}{2}\left(\Pr\left[\mathrm{P}_A^{\Pi_{\mathrm{ABS}}}(\lambda,1)=1\right] - \Pr\left[\mathrm{P}_A^{\Pi_{\mathrm{ABS}}}(\lambda,0)=1\right]\right) + \frac{1}{2} \\
&= \frac{1}{2}\mathrm{Adv}_A^{\mathrm{PP}}(\lambda) + \frac{1}{2}
\end{aligned}$$

In the other case $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda,b=0)$, the signatures are generated independently from the bit d that D chooses. The signature oracle $\mathcal{O}^{\mathrm{Sign}_0}_{\mathsf{pp},\mathsf{msk}}$ generates signatures with $\mathrm{SimSign}(\mathsf{pp},\mathsf{msk},m,\mathbb{P})$. SimSign generates the signature in both cases ($d=0$, $d=1$) only with the public parameters, master secret, message and policy as input. Hence, independent of the secret key for the attribute set $\mathbb{A}_d^*$. Consequently, the view of A is independent of d. Therefore, $\Pr[\tilde{b}=1] = \Pr[d=\tilde{d}] = \frac{1}{2}$. Consequently, it holds that $\mathrm{Adv}_D^{\mathrm{PSimP}}(\lambda) = \frac{1}{2}\mathrm{Adv}_A^{\mathrm{PP}}(\lambda)$. □

The following corollary originally shown in [BF14] follows from Theorem 2.1 and that the reduction given in the proof of Theorem 2.1 is efficient.

Corollary 2.1. *An ABS scheme* Π_{ABS} *that is computationally simulation private (Definition 2.9) is also computationally private (Definition 2.7).*

For policy-based signatures the authors show in [BF14] that without restricting the polices the reverse direction of Corollary 2.1 does not hold, by presenting a counterexample. Since, ABS is a special variant of policy-based signatures, the counterexample can also be applied to the reverse direction of Theorem 2.1. In the following we will restate the counterexample. To see that simulation-based privacy is a stronger notion than standard privacy, let us consider our example scheme above, where the Sign algorithms appends the used attribute set to the signature. Assume a policy class where computing a satisfying attribute set is computationally hard (satisfiability of CNF formulas). Obviously, the example scheme with this policy class is not perfectly simulation private, since under usual assumptions there is no probabilistic polynomial-time algorithm SimSign that given just the master secret and message, computes the satisfying attribute set and appends it to the signature.

2.2 On the Security of Existing Schemes

In Section 3 we present a wrapper protocol that transforms an ABS scheme to be UC compatible. Our main result Theorem 4.1 shows that if and only if the wrapped ABS scheme is correct, consistent, unforgeable, and simulation private then the wrapper protocol achieves UC security.

In the following we show for two existing ABS schemes [MPR11; SAH16] that they satisfy our perfect simulation privacy definition. Therefore, our main theorem (Theorem 4.1) shows that the two schemes can be used in the wrapper protocol to achieve UC security. Both schemes were originally shown to satisfy correctness, consistency, unforgeability, and the weaker notion perfect privacy. To show that an ABS schemes is perfectly simulation private we have to define three algorithms (SimSetup, SimKeyGen, SimSign) and prove that they satisfy Definition 2.9. To achieve this for the ABS schemes [MPR11; SAH16] we only apply minor modifications to the signature algorithms to define simulation signature algorithms SimSign. In particular these modifications do not imply changes to the setup and key generation algorithms.

Both ABS schemes [MPR11; SAH16] are generic constructions. To define a SimSign for the schemes, we exploit a commonality in the normal signature algorithms of both schemes. The normal signature algorithms fulfill two basic properties. First, the signature on a message-policy pair proves that the signer knows a valid secret key on attributes satisfying the policy. Second, it binds the policy to the signed message. In detail for [MPR11; SAH16], a signature on a message-policy pair $(m, \mathbb{P})$ of the normal signature algorithm Sign, proves that the signer knows a secret key for attributes satisfying the given policy $\mathbb{P}$ *OR* that he knows a special signature on $(m, \mathbb{P})$. We exploit the second property to define the SimSign algorithms for both schemes [MPR11; SAH16].

2.2.1 Generic ABS Construction by Sakai et al. [SAH16]

The SimSign algorithm that we present is implicitly given in the unforgeability proof in [SAH16] (Theorem 1 Game 2). For simplicity we just give a high-level description of the algorithm. The scheme [SAH16] uses a non-interactive perfect witness indistinguishable (NIWI) proof system, a

collision-resistant hash function H, a secure structure-preserving signature scheme and supports circuits C as policies. For further details especially for the NIWI proofs we refer to [SAH16]. Basically, SimSign is the original signature algorithm Sign of the ABS scheme [SAH16] executed with a special secret key generated on the hash value of the message-policy pair (m, C). This can only be done with the master secret.

SimSign $(\mathsf{pp}, \mathsf{msk}, m, C)$ On input public parameter pp, master secret msk, message m and policy C (policy is a circuit): Step 1 of Sign is modified such that it first computes a special secret key sk_{x^*}. Therefore, it first computes $h \leftarrow \mathrm{H}(\mathsf{hk}, \langle m, C \rangle)$, sets $h := (h_1 \parallel \ldots \parallel h_{l_\mathrm{H}})$ and $x^* := (g^1, g^{h_1}, \ldots, g^{h_{l_\mathrm{H}}}, 1, \ldots, 1) \in \mathbb{G}_1^{l+1}$. Then it computes a structure-preserving signature θ^* on x^* and sets $\mathsf{sk}_{x^*} := (x^*, \theta^*)$. Then it expands the circuit C to $\hat{C}$ as in Sign, goes on with steps $2-10$ of Sign and generates a NIWI proof π with θ^* as the witness. At the end of step 10 it outputs signature $\sigma := \pi$.

Changes to Setup and KeyGen are not necessary, since SimSign uses the master secret msk which is already output by Setup. Therefore, we use Setup as SimSetup and KeyGen as SimKeyGen.

Theorem 2.2. Π_{Sakai} *from [SAH16] combined with* $(\mathrm{SimSetup}, \mathrm{SimKeyGen}, \mathrm{SimSign})$ *as defined above is perfectly simulation private (Definition 2.9).*

Proof. Since SimSetup is Setup and SimKeyGen is KeyGen their distribution is unchanged. What is left to show is, that SimSign generates the same distribution as Sign. The output of SimSign and Sign is a proof π that was generated by a perfectly witness indistinguishable proof system. Sign uses the secret key of the signer as the witness, whereas SimSign generates a special secret key x^*, θ^* on-demand with the help of the master secret msk and uses this secret key as the witness. Notice, that both algorithms expand the given circuit C such that it is also satisfied by the hash value used to compute the special secret key. From the perfect witness indistinguishability of the proof system it follows that the distributions of SimSign and Sign are equivalent. □

2.2.2 Generic ABS Construction by Maji et al. [MPR11]

Maji, Prabhakaran and Rosulek present in [MPR11] a generic construction of ABS supporting monotone boolean functions as policies. The construction is based on a NIWI proof systems and so called credential bundles. A secure credential bundle $\mathrm{CB} = \{\mathrm{Setup}, \mathrm{Gen}, \mathrm{Ver}\}$, in its simplest from, can be instantiated with a secure digital signature scheme, cf. [MPR11]. The approach of the SimSign algorithm for this construction is similar to the one that we used above for [SAH16]. Again it is possible to generate a special element using the master secret. The scheme [MPR11] supports a universe of attributes that contains a pseudo-attribute for every message-policy pair $(m, \mathbb{P})$. The scheme in [MPR11] defines a normal signature algorithm Sign that on input $(m, \mathbb{P})$ and a secret key sk first extends the policy to $\mathbb{P}' := \mathbb{P} \vee$ "pseudo-attribute $(m, \mathbb{P})$". Second, it uses the signers secret key sk to generate the signature. For the simulation signature algorithm SimSign we use the master secret to issue a special secret key sk_* for the pseudo-attribute $(m, \mathbb{P})$ and use the extended $\mathbb{P}'$ to generate a signature in the same way as Sign generates it.

First we adapt the syntax and then present the SimSign algorithm in detail. The generic ABS construction in [MPR11] describes five algorithms TSetup, ASetup, AttrGen, Sign and

Verify. They define two setup algorithms to separate the setup of the NIWI proofs from the setup for key generation. We define Setup such that it combines TSetup and ASetup in one algorithm. Further, let KeyGen run AttrGen. Accordingly, we define the ABS scheme as $\Pi_{\text{Maji}} = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify})$. Let $\mathbb{U}$ be the universe of attributes. Let $\mathbb{U}'$ denote the space of pseudo-attributes, where $\mathbb{U} \cap \mathbb{U}' = \emptyset$. One pseudo-attribute $a_{m,\Upsilon} \in \mathbb{U}'$ is added for each pair of message m and policy Υ. A policy is defined as a monotone boolean function $\Upsilon\colon 2^{\mathbb{U}} \to \{0,1\}$. For perfect simulation privacy we have to provide a SimSetup, SimKeyGen, and SimSign algorithm. We use Setup as SimSetup and KeyGen as SimKeyGen. The following SimSign algorithm is partially given in the proof of Theorem 1 in the original work [MPR11].

SimSign $(\mathsf{pp}, \text{tsk}, m, \Upsilon)$ On input public parameter pp, master secret tsk, message m and policy Υ: Generate a special secret key sk_* with the pseudo-attribute $a_{m,\Upsilon}$ for (m, Υ), $\mathsf{sk}_* \coloneqq (\tau, \varsigma) \leftarrow \text{CB.Gen}(\text{tsk}, a_{m,\Upsilon})$ output $\pi \leftarrow \text{Sign}(\mathsf{pp}, sk_*, m, \Upsilon)$.

Theorem 2.3. Π_{Maji} *from [MPR11] combined with* $(\text{SimSetup}, \text{SimKeyGen}, \text{SimSign})$ *as defined above is perfectly simulation private (Definition 2.9).*

The signatures generated by Sign and SimSign are NIWI proofs with different witnesses. Hence, as in Theorem 2.2 the above theorem follows from the perfect witness indistinguishability of the proof system.

3 Universal Composable Attribute-based Signature Schemes

We first give a brief introduction to Canetti's Universal Composability framework [Can01] and responsive environments [Cam+16]. Based on this we present our universally composable ideal functionality for attribute-based signatures. After that we introduce a wrapper protocol that transforms ABS schemes into an UC compatible formulation. In Section 4 we show that the wrapper protocol is a secure UC-realization of our ideal ABS functionality if and only if the wrapped ABS scheme is correct, consistent, unforgeable and computationally simulation private.

3.1 Preliminaries: Universal Composability Framework

We briefly summarize the important parts of the Universal Composability framework [Can01]. For a detailed description we refer to [Can13]. In the UC framework a task or a scheme is described as an ideal functionality $\mathcal{F}$ and compared to a execution of a protocol π. The protocol π involves an adversary and parties that handle their tasks on their own. Each of the parties runs an instance of the protocol. The ideal functionality models a trusted party that handles all tasks in the name of all other parties. Here the parties are just dummies. In both settings, there is an environment $\mathcal{E}$ that controls the input and activations of parties. The environment is considered as the execution environment of the tasks, since in reality protocols do not exist in a vacuum. The UC security is based on a simulation paradigm. Protocol π is called UC secure, if for every adversary $\mathcal{A}$ against the protocol π, there exists a simulator $\mathcal{S}$ considered as the ideal adversary against $\mathcal{F}$, such that no environment $\mathcal{E}$ can distinguish if it talks to $\mathcal{F}$ with $\mathcal{S}$ or to π with $\mathcal{A}$. To be more precise, let the random variable $\mathcal{E}[\pi, \mathcal{A}](\lambda, z)$ denote the output of $\mathcal{E}$ when interacting with π

and $\mathcal{A}$ with security parameter $\lambda \in \mathbb{N}$ on input $z \in \{0,1\}^*$. Accordingly, let $\mathcal{E}\left[\mathcal{F}, \mathcal{S}^{\mathcal{A}}\right](\lambda, z)$ be the random variable that denotes the output of $\mathcal{E}$ when interacting with $\mathcal{F}$ and $\mathcal{S}^{\mathcal{A}}$ with $\lambda \in \mathbb{N}$ on input $z \in \{0,1\}^*$, where $\mathcal{S}^{\mathcal{A}}$ denotes black-box access to adversary $\mathcal{A}$. The former random variable is defined over the randomness of all parties and $\mathcal{A}$ and the latter is defined over the randomness of $\mathcal{S}^{\mathcal{A}}$. For an ideal functionality $\mathcal{F}$ and protocol π, we say π realizes $\mathcal{F}$ (short π is UC secure) if for all adversaries $\mathcal{A}$, there is a simulator $\mathcal{S}^{\mathcal{A}}$ such that, for all environments $\mathcal{E}$ and all $z \in \{0,1\}^*$, $\mathcal{E}\left[\mathcal{F}, \mathcal{S}^{\mathcal{A}}\right](\lambda, z)$ and $\mathcal{E}\left[\pi, \mathcal{A}\right](\lambda, z)$ are computationally indistinguishable in λ. In the following we assume that $\mathcal{E}$, $\mathcal{A}$, $\mathcal{S}^{(\cdot)}$ are probabilistic polynomial-time algorithms with respect to the security parameter λ. In our notation we omit the security parameter λ and input z. We refer to an interaction of $\mathcal{E}$ with $[\mathcal{F}, \mathcal{S}^{(\cdot)}]$ as the ideal setting and with $[\pi, \mathcal{A}]$ as the real setting. For simplicity one can always assume the so called standard (dummy) adversary $\mathcal{A}$, that forwards every message received with no delay and is inactive apart from that. Canetti [Can01] showed that the standard adversary is the strongest adversary in UC, since $\mathcal{E}$ takes over the responsibilities of $\mathcal{A}$.

We consider adaptive corruption in the sense that at any time the adversary $\mathcal{A}$ can trigger a corruption of a party. $\mathcal{A}$ is in full control of a corrupted party. We restrict the corruption such that the party responsible for the setup of the scheme can only be corrupted after an honest setup was completed. This restriction is denoted as adaptive* and is explained in more detail in Section 4. Further, we consider the erasure model, where honest parties erase ephemeral randomness immediately after usage. This means that on corruption the adversary $\mathcal{A}$ only gets the result of previous computations but not the randomness used. For communication between the parties to send secret information such as secret keys, we rely on secure channels. Accordingly our results are shown in an adaptive*, erasure, secure channels model, denoted as $\mathcal{F}$ [adaptive*, erasure, secure channels].

In UC the ideal functionality and simulator often exchange some meta-data for modeling reasons. Typically protocol designers assume that the simulator answers immediately where according to the UC framework the simulator can do anything he likes after receiving a message from the ideal functionality. This introduces not anticipated state changes of parties and unexpected behavior of the ideal functionality. Therefore, the handling of the exchanged data, and responses of the simulator makes the description of the ideal functionality more complex for protocol designers. Then again the side effects of meta-data exchanges should not be ignored as Camenisch et al. highlight in [Cam+16]. To circumvent the problems the authors [Cam+16] define responsive environments where the adversary and the environment are bound to answer to so called restricting messages immediately. We model or ideal functionality for responsive environments. Therefore, we employ the generic restriction defined by Camenisch et al. [Cam+16] where every message prefixed with "Respond" is restricting.

3.2 Ideal ABS Functionality

The ideal functionality for ABS models a scheme with an unique setup party P_{Setup}. After a Setup activation through P_{Setup} was completed any party P can ask for a secret key. All activations are instantiated by the environment $\mathcal{E}$. The activation to query a secret key, given an attribute set, is called Key Generation. A signing party can generate a signature on a message

and a policy by a Signature activation and any party can verify a signature through a Verify activation.

In Figure 1 the ideal ABS functionality $\mathcal{F}_{\text{ABS}}$ with access to a simulator $\mathcal{S}$ is given. The algorithms S.Setup, S.KG, S.Sign and S.Verify output by $\mathcal{S}$ are stateless ppt algorithms.

3.2.1 Description of $\mathcal{F}_{\text{ABS}}$

We describe each activation of $\mathcal{F}_{\text{ABS}}$ (Figure 1) in detail. After that we will concentrate on the security guarantees of $\mathcal{F}_{\text{ABS}}$. We start with the explanation of general UC mechanisms. In the UC framework each ideal functionality instance has a unique session id sid. The sid can be determined in several ways (cf. [Can13], Section 3.1.3). For simplicity we let the first Setup activation determine the sid, which consists of the party's identifier and a unique session identifier sid$'$. Therefore, the sid also determines the unique party responsible for the setup and key generation. An instance of the functionality stores the sid in the first Setup activation and ignores all activations with a different sid$'$. The inputs for the Setup, Key Generation, Signature and Verify activations are determined by the environment $\mathcal{E}$. Note, messages prefixed with "Respond" are answered immediately by the simulator [Cam+16] and a public delayed output is an output to a party that first has to be acknowledged by the simulator.

Setup: In the Setup activation the simulator $\mathcal{S}$ is responsible for providing ppt algorithms S.Setup, S.KG, S.Sign, and S.Verify. These have to be stateless ppt algorithms such that the outputs of $\mathcal{F}_{\text{ABS}}$ generated with these algorithms are independent of the internal state of the simulator and previous activations. Further, this modeling allows us to give a technically sound equivalence proof in Section 4). The algorithm S.Setup is used to generate and fix the public parameters pp and the master secret msk of the functionality instance. The algorithm S.KG and S.Sign always take as input the recorded $(\mathsf{pp}, \mathsf{msk})$. They are only given as an explicit input to highlight that we use the recorded pair.

Key Generation: This activation models an exchange between a party P_i that just queries a key on a given attribute set (KeyGenRequest) and the party P_{Setup} responsible for the actual key generation (KeyGen). The activation for P_{Setup} models that it is triggered after receiving the output $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}, P_i)$ from the activation of P_i; telling the setup party that P_i asks for a key on $\mathbb{A}$. This modeling also encompasses that P_{Setup} can decide whether to answer a key generation request. P_{Setup} proceeds only if there is an unprocessed key generation request with the key identifier kid for the party P_i. For honest P_{Setup} the ideal functionality $\mathcal{F}_{\text{ABS}}$ first checks that the attribute set is valid and then it uses the algorithm S.KG provided by $\mathcal{S}$ to generate the secret key for $\mathbb{A}$. Next, $\mathcal{S}$ is informed about the key generation. This is necessary since $\mathcal{S}$ has to simulate the transmission of the secret key to the party P_i. If the final output to P_i is delivered, $\mathcal{F}_{\text{ABS}}$ is assured that the simulator transmitted the secret key and $\mathcal{F}_{\text{ABS}}$ can record a successful key generation for P_i. In the case of corrupted P_{Setup}, $\mathcal{F}_{\text{ABS}}$ records a key generation request ($\perp$ for unknown). Therefore, in the Signature activation $\mathcal{F}_{\text{ABS}}$ can ask the simulator $\mathcal{S}$ to sign the message. This is delegated to $\mathcal{S}$ since if P_{Setup} is corrupt $\mathcal{F}_{\text{ABS}}$ can not record the generated secret keys.

Setup On input $(\text{Setup}, \text{sid})$ from a party P_{Setup}

1. If $\text{sid} = (P_{\text{Setup}}, \text{sid}')$ for some sid' continue, else ignore.
2. If there is no record $(\text{sid}', P_{\text{Setup}}, \mathsf{pp}, \mathsf{msk})$, send $(\text{Respond}, (\text{Setup}, \text{sid}))$ to $\mathcal{S}$. Upon receiving $(\text{Setup}, \text{sid}, \text{S.Setup}, \text{S.KG}, \text{S.Sign}, \text{S.Verify})$ from $\mathcal{S}$ do:
   a) Generate $(\mathsf{pp}, \mathsf{msk}) \leftarrow \text{S.Setup}$
   b) Record $(\text{sid}', P_{\text{Setup}}, \mathsf{pp}, \mathsf{msk})$ and $(\text{S.Setup}, \text{S.KG}, \text{S.Sign}, \text{S.Verify})$.
3. Else, use recorded $(\text{sid}', P_{\text{Setup}}, \mathsf{pp}, \mathsf{msk})$.
4. In both cases output $(\text{Public Params}, \text{sid}, \mathsf{pp})$ to P_{Setup}.

Key Generation On input $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A})$ from party P_i

1. If $\text{sid} = (P_{\text{Setup}}, \text{sid}')$ continue, else ignore. Mark the request as unprocessed.
2. Send public delayed output $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}, P_i)$ to P_{Setup}

Key Generation On input $(\text{KeyGen}, \text{sid}, \text{kid})$ from party P_{Setup}

1. If $\text{sid} = (P_{\text{Setup}}, \text{sid}')$ and there is an unprocessed $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A})$ from party P_i, mark it processed and continue. Else ignore.
2. If P_{Setup} is corrupt then record $(\text{KeyGen}, \text{sid}, \text{kid}, \perp, \perp, \mathbb{A}, \perp)$ and send as a public delayed output $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A})$ to P_i.
3. Else, check $\mathbb{A} \subseteq \mathbb{U}(\mathsf{pp})$ for recorded $(\text{sid}', P_{\text{Setup}}, \mathsf{pp}, \mathsf{msk})$, if not ignore.
   a) Generate secret key $\mathsf{sk}_{\mathbb{A}} \leftarrow \text{S.KG}(\mathsf{pp}, \mathsf{msk}, \mathbb{A})$ and record $(\text{KeyGen}, \text{sid}, \text{kid}, \perp, \mathsf{pp}, \mathbb{A}, \mathsf{sk}_{\mathbb{A}})$. Send $(\text{Respond}, \text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A}, P_i, \mathsf{sk}_{\mathbb{A}})$ to $\mathcal{S}$.
   b) Upon receiving $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A}, P_i, \mathsf{sk}_{\mathbb{A}}, 1)$ from $\mathcal{S}$ send $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A})$ as a public delayed output to P_i.
4. When $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A})$ is delivered to P_i update the record by adding the party to $(\text{KeyGen}, \text{sid}, \text{kid}, P_i, \cdot, \mathbb{A}, \cdot)$

Signature On input $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', \mathbb{A}, m, \mathbb{P})$ from some party P_i

1. If $\text{sid} = (P_{\text{Setup}}, \text{sid}')$, $\mathbb{A} \subseteq \mathbb{U}(\mathsf{pp}')$ and $\mathbb{P} \in \mathbb{U}(\mathsf{pp}')$ continue, else ignore.
2. Check for $(\text{KeyGen}, \text{sid}, \text{kid}, P_i, \mathsf{pp}', \mathbb{A}, \cdot)$ where $\mathbb{P}(\mathbb{A}) = 1$, if not ignore.
3. If $\mathsf{pp}' = \mathsf{pp}$, run $\text{S.Sign}(\mathsf{pp}, \mathsf{msk}, m, \mathbb{P})$ and get σ. (guarantees privacy)
4. Else, send $(\text{Respond}, (\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', \mathbb{A}, m, \mathbb{P}))$ to $\mathcal{S}$ and receive the answer $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', \mathbb{A}, m, \mathbb{P}, \sigma)$ from $\mathcal{S}$.
5. If a record $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', m, \mathbb{P}, \sigma, 0)$ exists, output error and halt.
6. Else, record $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', m, \mathbb{P}, \sigma, 1)$ and output $(\text{Signature}, \text{sid}, \text{kid}, \mathbb{A}, m, \mathbb{P}, \sigma)$ to P_i.

Verify On input $(\text{Verify}, \text{sid}, \mathsf{pp}', m, \mathbb{P}, \sigma)$ from some party P

1. Get $b \leftarrow \text{S.Verify}(\mathsf{pp}', m, \mathbb{P}, \sigma)$
   I. If $\mathsf{pp}' = \mathsf{pp}$ for recorded pp then
      i. If $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}, m, \mathbb{P}, \sigma, f)$ recorded for any f, then set $f_{\text{out}} := f$ (guarantees correctness/consistency)
      ii. Else, if for any σ' entry $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}, m, \mathbb{P}, \sigma', 1)$ exists or P_{Setup} is corrupt or there exists at least one corrupted P_i with a record $(\text{KeyGen}, \text{sid}, \text{kid}, P_i, \mathsf{pp}, \mathbb{A}', \cdot)$ where $\mathbb{P}(\mathbb{A}') = 1$, then set $f_{\text{out}} := b$ and record $(\text{Signature}, \text{sid}, \perp, \mathsf{pp}, m, \mathbb{P}, \sigma, f_{\text{out}})$
      iii. Else, set bit $f_{\text{out}} := 0$ and record $(\text{Signature}, \text{sid}, \perp, \mathsf{pp}, m, \mathbb{P}, \sigma, 0)$. (guarantees unforgeability)
   II. If $\mathsf{pp}' \neq \mathsf{pp}$ for recorded pp then
      i. If $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', m, \mathbb{P}, \sigma, f)$ is recorded, set $f_{\text{out}} := f$
      ii. Else $f_{\text{out}} := b$ and record $(\text{Signature}, \text{sid}, \perp, \mathsf{pp}', m, \mathbb{P}, \sigma, f_{\text{out}})$
2. Output $(\text{Verified}, \text{sid}, m, \mathbb{P}, \sigma, f_{\text{out}})$ to P

Figure 1: Attribute-Based Signature Ideal Functionality $\mathcal{F}_{\text{ABS}}$

Signature: After a check if the inputs are valid the Signature activation checks if the activated party has a satisfying secret key for the given policy. If the activation is under registered pp, it utilizes S.Sign to output a signature without using the secret key of the party and without the activated party's identity as an input to S.Sign. Otherwise it asks $\mathcal{S}$ for a signature under unregistered public parameters. In general, each signature generated in $\mathcal{F}_{\text{ABS}}$ is recorded as valid.

Verify: In this activation we handle any public parameters. Hence, we cover the cases where pp' is invalid or belongs to another instance, even if the activated party is honest. The simplest verify case is if there is no corrupt party. Then $\mathcal{F}_{\text{ABS}}$ verifies all signatures where the corresponding message and policy was not signed in a Signature activation as invalid (Verify I.iii, $f_{\text{out}} \coloneqq 0$). Verify step I.ii handles two cases. First, a presumably manipulated signature (e.g. randomized) that was not recorded in a Signature activation. Second, the existence of corrupted parties. In any case, to decide whether a signature is valid we use the algorithm S.Verify, provided by $\mathcal{S}$, and the results is recorded by $\mathcal{F}_{\text{ABS}}$.

Every party can be corrupted in our adaptive* model, with the restriction that the setup party P_{Setup} can only be corrupted after a finished Setup activation. We explain this in more detail in Section 4. On a high level and assuming the standard adversary $\mathcal{A}$, the environment can trigger a corruption of any party P by sending $(\mathit{corrupt}, P)$. In case of a corruption of P_{Setup}, $\mathcal{E}$ gets the master secret stored in P_{Setup}. Hence, $\mathcal{E}$ can issue arbitrary secret keys. This includes the issuing of secret keys to honest parties. For this case, $\mathcal{F}_{\text{ABS}}$ also keeps records of successful key generations with an honest party P_i and corrupted P_{Setup}. Additionally, $\mathcal{F}_{\text{ABS}}$ only generates signatures for P_i if such a record exists.

3.2.2 Security of $\mathcal{F}_{\text{ABS}}$

In the following we explain the security guarantees modeled by $\mathcal{F}_{\text{ABS}}$. This includes the description of the scope of the guarantees and how the keys are managed.

Scope of Security: The signatures functionality by Canetti [Can03] supports the verification under any public key. This allows the functionality to be more modular and to be used in different applications. $\mathcal{F}_{\text{ABS}}$ is based on that idea and supports Signature and Verify activations with unregistered public parameters pp'. $\mathcal{F}_{\text{ABS}}$ only guarantees security under the registered public parameters. Therefore we have to check in Key Generation, Signature and Verify if the given public parameters pp' are equal to the registered public parameters pp. For unregistered public parameters in the Signature activation (step 4) we ask the simulator $\mathcal{S}$ or we rely on the algorithm given by $\mathcal{S}$ in the Verify activation (step 1). In this case the guarantees are determined by $\mathcal{S}$. We also allow the corruption of any party with the restriction that P_{Setup} can only be corrupted after the Setup activation was executed once.

Correctness/Consistency: For an honest signer, a Signature activation like $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', \mathbb{A}, m, \mathbb{P})$ always records $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', m, \mathbb{P}, \sigma, f \coloneqq 1)$. This leads to a verification output with $f_{\text{out}} = 1$ in a corresponding Verify activation (step I.i). Thus, correctness

is guaranteed. Consistency is captured by the steps I.i and II.i. There we just output what is recorded. To verify $(m, \mathbb{P}, \sigma)$ where $\mathcal{F}_{\text{ABS}}$ already generated a different signature σ' for $(m, \mathbb{P})$ we employ step I.ii and use the bit output by S.Verify. The step I.ii also handles the case of corrupted parties. Corrupted parties generate signatures without the involvement of $\mathcal{F}_{\text{ABS}}$ and may share their secret keys. In these cases, we have to use the output b of S.Verify. Hence, the guarantees are those provided by the simulator $\mathcal{S}$ and the algorithms that $\mathcal{S}$ outputs.

Non Colluding: Is handled in the Signature activation step 2. There we check if the activated party has a single secret key for attributes that satisfy the given policy.

Privacy: Privacy for honest users under registered public parameters is guaranteed by the input restriction on the algorithms returned by $\mathcal{S}$. In particular, the S.Sign generates signatures with the public parameters, the master secret, a message, and a policy as input. Therefore it generates signatures without the parties secret key or attributes as an input. Hence, for honest signing parties and under the registered pp we guarantee that the signatures output by $\mathcal{F}_{\text{ABS}}$ are independent of the attribute sets encoded in the secret keys of the signing parties. Therefore, the signatures can not be linked to a party, an attribute set, and to a secret key by an adversary. Since, $\mathcal{F}_{\text{ABS}}$ only guarantees privacy under the registered public parameters pp, we can ask $\mathcal{S}$ for a signature under unregistered public parameters pp' or if P_{Setup} was already corrupted during the corresponding key generation (Signature activation step 4). In a Key Generation activation with honest P_{Setup} we guarantee, that the secret key is independent of the party's identifier. Therefore, we use the S.KG algorithm where the identifier P_i is not an input. Even if P_{Setup} is corrupted after a successful key generation with party P_i, we require that a signature can be generated without the knowledge of P_i's secret key by using S.Sign with the registered $(\mathsf{pp}, \mathsf{msk})$ pair.

Unforgeability: It is guaranteed through the Verify step I.iii. Here we set the bit f_{out} to 0, if all parties are honest and we have checked that the corresponding message and policy was not signed by $\mathcal{F}_{\text{ABS}}$ in a Signature activation.

3.3 Protocol

We present our ABS wrapper protocol π_{ABS} in Figure 2. It serves as a generic transformation of an ABS scheme Π_{ABS} to the UC framework. The protocol activations Setup, Key Generation, Signature and Verify use the algorithms of Π_{ABS} as a black-box. We will show that the protocol π_{ABS} realizes the ideal functionality $\mathcal{F}_{\text{ABS}}$. The Setup activation of the protocol is run by a unique party with the identifier P_{Setup}, it also runs the corresponding side of the Key Generation. The other side of Key Generation as well as the Signature and Verify protocol parts can be executed by any party. Each party runs an instance of the protocol. For example, a party P that is triggered by the environment and wants to generate a signature, executes the Signature protocol part, given in Figure 2. Notice, secret keys are never output to $\mathcal{E}$ in the protocol π_{ABS}, since in a real world application a honest party should never reveal its secret keys.

Setup When party P_{Setup} receives first input $(\text{Setup}, \text{sid})$ from $\mathcal{E}$

1. If $\text{sid} = (P_{\text{Setup}}, \text{sid}')$ for some sid' continue, else ignore.
2. P_{Setup} runs $(\mathsf{pp}, \mathsf{msk}) \leftarrow \text{Setup}\left(1^{\lambda}\right)$ and records $(\text{sid}', P_{\text{Setup}}, \mathsf{pp}, \mathsf{msk})$.
3. Outputs $(\text{Public Params}, \text{sid}, \mathsf{pp})$ to $\mathcal{E}$.

Key Generation

When party P_i receives $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A})$ from $\mathcal{E}$

1. If sid has the form $(P_{\text{Setup}}, \text{sid}')$ continue, else ignore.
2. P_i sends $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A})$ to P_{Setup}.
3. On receiving $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A})$, P_{Setup} records request from P_i and outputs $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}, P_i)$ to $\mathcal{E}$.
4. On receiving $(\text{KeyGen}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}, \mathsf{sk}_{\mathbb{A}})$ as an answer from P_{Setup}
   a) If $(\mathsf{sk}_{\mathbb{A}}, \mathbb{A})$ is valid under received pp, P_i appends record $(\text{KeyGen}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}, \mathsf{sk}_{\mathbb{A}})$ and outputs $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A})$ to $\mathcal{E}$.

When party P_{Setup} receives $(\text{KeyGen}, \text{sid}, \text{kid})$ from $\mathcal{E}$

1. If $\text{sid} = (P_{\text{Setup}}, \text{sid}')$ continue, else ignore.
2. Look up request record $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A})$ from P_i.
3. If $(\text{sid}', P_{\text{Setup}}, \mathsf{pp}, \mathsf{msk})$ is not recorded by P_{Setup} or $\mathbb{A} \nsubseteq \mathbb{U}(\mathsf{pp})$ ignore.
4. Else P_{Setup} computes $\mathsf{sk}_{\mathbb{A}} \leftarrow \text{KeyGen}(\mathsf{pp}, \mathsf{msk}, \mathbb{A})$ and sends $(\text{KeyGen}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}, \mathsf{sk}_{\mathbb{A}})$ to P_i.

Signature When party P_i receives $(\text{Signature}, \text{sid}, \text{kid}, \mathsf{pp}', \mathbb{A}, m, \mathbb{P})$ from $\mathcal{E}$

1. If $\text{sid} = (P_{\text{Setup}}, \text{sid}')$, $\mathbb{A} \subseteq \mathbb{U}(\mathsf{pp}')$ and $\mathbb{P} \in \mathbb{U}(\mathsf{pp}')$ continue, else ignore;
2. P_i looks up last record $(\text{KeyGen}, \text{sid}, \text{kid}, \mathsf{pp}', \mathbb{A}, \mathsf{sk}_{\mathbb{A}})$ where $\mathbb{P}(\mathbb{A}) = 1$.
3. If there is no record, then P_i ignores the activation.
4. Else, P_i computes signature $\sigma \leftarrow \text{Sign}(\mathsf{pp}', \mathsf{sk}_{\mathbb{A}}, m, \mathbb{P})$ and outputs $(\text{Signature}, \text{sid}, \text{kid}, \mathbb{A}, m, \mathbb{P}, \sigma)$ to $\mathcal{E}$

Verify When party P_i receives $(\text{Verify}, \text{sid}, \mathsf{pp}', m, \mathbb{P}, \sigma)$ from $\mathcal{E}$

1. P_i runs $b \leftarrow \text{Verify}(\mathsf{pp}', m, \mathbb{P}, \sigma)$ and outputs $(\text{Verified}, \text{sid}, m, \mathbb{P}, \sigma, b)$

Figure 2: Attribute-based Signature Protocol π_{ABS}

4 Security

The UC-realization is proven under the assumption that the setup party can only be corrupted after the setup activation was executed once. This is a restriction on the UC environment and is denoted as adaptive*. It models the guarantees of our privacy definitions (Definition 2.7 and 2.9) and mirrors existing experiment-based secure ABS schemes that are defined with an honest setup [AAS16; EHM11; MPR08; MPR11; OT14; SAH16]. The restrictions can be avoided by defining ABS in the CRS model, similar to the blind signatures by Abe and Ohkubo [AO12].

Similar environment restrictions regarding the setup of schemes are present in other UC-realizations. In the work of Buan et al. [BGK06] the signer also can only be corrupted after the key generation. For UC non-committing blind signatures [AO12] the environment is restricted to activate the key generation only once.

Theorem 4.1. *ABS scheme* Π_{ABS} *is correct, consistent, unforgeable and computationally simulation private if and only if* π_{ABS} *realizes* $\mathcal{F}_{\mathrm{ABS}}$[*adaptive*,erasure, secure channels*].

We split the theorem in two parts and proof them separately. To shorten notation we will omit the assumptions [adaptive*, erasure, secure channels] in textual descriptions.

4.1 Experiment-Based Security implies UC Security

In this section we show that an correct, consistent, unforgeable, and computationally simulation private attribute-based signature scheme Π_{ABS} is also UC secure. The UC security is shown with respect to our ideal ABS functionality $\mathcal{F}_{\mathrm{ABS}}$ (Figure 1) and the ABS protocol π_{ABS} (Figure 2) which is a wrapper for an ABS scheme Π_{ABS}.

Theorem 4.2. *If* Π_{ABS} *is correct, consistent, unforgeable and comp. simulation private, then* $\mathcal{F}_{\mathrm{ABS}}$ [*adaptive*, erasure, secure channels*] *is realized by* π_{ABS}.

In the following we define one simulator $\mathcal{S}^{(\cdot)}$ that, for any given adversary $\mathcal{A}$, interacts with $\mathcal{F}_{\mathrm{ABS}}$ such that for all environments $\mathcal{E}$, it holds that $[\mathcal{F}_{\mathrm{ABS}}, \mathcal{S}_0^{\mathcal{A}}]$ and $[\pi_{\mathrm{ABS}}, \mathcal{A}]$ are indistinguishable. For an ABS scheme $\Pi_{\mathrm{ABS}} = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify}, \mathrm{SimKeyGen}, \mathrm{SimSetup}, \mathrm{SimSign})$, any adversary $\mathcal{A}$ and $\mathcal{F}_{\mathrm{ABS}}$ we define the simulator $\mathcal{S}_0^{\mathcal{A}}$ in Figure 3, where $\mathcal{S}_0^{\mathcal{A}}$ has black-box access to $\mathcal{A}$. In case of a corruption of P_{Setup}, $\mathcal{E}$ gets the master secret stored in P_{Setup}. Hence, $\mathcal{E}$ can issue arbitrary secret keys. This includes the issuing of secret keys to honest parties. If a corrupted party interacts with a honest party the interaction is handled by $\mathcal{S}_0^{\mathcal{A}}$ as described in Figure 3.

The proof of Theorem 4.2 is done through a sequence of games. Starting in Game 0 with the ideal setting and ending in Game 5 with the real setting. In particular, if $\mathcal{E}$ can distinguish the ideal setting and the real setting then it has to distinguish one of the consecutive games. In the sequence of games we gradually modify $\mathcal{F}_{\mathrm{ABS}}$ and $\mathcal{S}_0^{\mathcal{A}}$ in each step. Thereby, every game expands the modifications introduced in the games before.

With each modification we show that if an environment $\mathcal{E}$ can distinguish Game i-1 and Game i, then there is an adversary that breaks one of the security guarantees of ABS.

In Figure 4 we present the sequence of games and the related security guarantees; computational simulation privacy, unforgeability, correctness and consistency.

Setup On input (Respond, (Setup, sid)) from $\mathcal{F}_{\text{ABS}}$

1. Send (Setup, sid, SimSetup(1^λ), SimKeyGen, SimSign, Verify) to $\mathcal{F}_{\text{ABS}}$

Key Generation: If P_i is honest

1. On seeing public delayed output (KeyGenRequest, sid, kid, pp, $\mathbb{A}$, P_i) for P_{Setup} by $\mathcal{F}_{\text{ABS}}$ send (KeyGenRequest, sid, kid, $\mathbb{A}$) from P_i to P_{Setup}

Key Generation: If P_{Setup} is honest

- On receiving message (Respond, (KeyGen, sid, kid, $\mathbb{A}$, P_i, $\mathsf{sk}_{\mathbb{A}}$)) from $\mathcal{F}_{\text{ABS}}$ record (KeyGen, sid, kid, pp, $\mathbb{A}$, P_i, $\mathsf{sk}_{\mathbb{A}}$) and send (KeyGen, sid, kid, $\mathbb{A}$, P_i, $\mathsf{sk}_{\mathbb{A}}$, 1) to $\mathcal{F}_{\text{ABS}}$.

1. On seeing the public delayed output (KeyGen, sid, kid, $\mathbb{A}$) for P_i by $\mathcal{F}_{\text{ABS}}$ simulate the P_{Setup} key generation part:
2. Check for the existence of record (KeyGen, sid, kid, pp, $\mathbb{A}$, P_i, $\mathsf{sk}_{\mathbb{A}}$). If it exists then send (KeyGen, sid, kid, pp, $\mathbb{A}$, $\mathsf{sk}_{\mathbb{A}}$) from P_{Setup} to P_i.

Key Generation: If P_i is corrupt

1. On receiving (KeyGenRequest, sid, kid, $\mathbb{A}$) sent to P_{Setup} from P_i (controlled by $\mathcal{A}$). Send (KeyGenRequest, sid, kid, $\mathbb{A}$) on behalf of P_i to $\mathcal{F}_{\text{ABS}}$.

Key Generation: If P_{Setup} is corrupt

1. On (KeyGen, sid, kid, pp', $\mathbb{A}$, sk) from P_{Setup} (controlled by $\mathcal{A}$). If (sk, $\mathbb{A}$) is valid under pp', send (KeyGen, sid, kid) on behalf of P_{Setup} to $\mathcal{F}_{\text{ABS}}$.
2. On seeing the public delayed output (KeyGen, sid, kid, $\mathbb{A}$) for P_i by $\mathcal{F}_{\text{ABS}}$ record (KeyGen, sid, kid, pp', $\mathbb{A}$, P_i, sk)

Signature On input (Respond, (Signature, sid, kid, pp', $\mathbb{A}$, m, $\mathbb{P}$)) from $\mathcal{F}_{\text{ABS}}$

1. Check for record (KeyGen, sid, kid, pp', $\mathbb{A}$, P_i, $\mathsf{sk}_{\mathbb{A}}$), if not ignore. Else, $\sigma \leftarrow$ Sign(pp', $\mathsf{sk}_{\mathbb{A}}$, m, $\mathbb{P}$) and send (Signature, sid, kid, pp', $\mathbb{A}$, m, $\mathbb{P}$, σ) to $\mathcal{F}_{\text{ABS}}$.

Others

- On (corrupt,P_i) from $\mathcal{A}$. Inform $\mathcal{F}_{\text{ABS}}$ with (corrupt,P_i), to get all input, output and exchanged messages including (KeyGen, sid, pp, $\mathbb{A}_j$, P_i) messages. Send it all to $\mathcal{A}$.
- Public delayed output send to honest parties is delivered after the steps are processed. For corrupted parties it is never delivered.

Figure 3: Simulator $\mathcal{S}_0^{\mathcal{A}}$

Game 0: Let $\mathcal{F}_{\text{ABS}}$ from Figure 1 be $\mathcal{F}_0$. $\mathcal{E}$ interacts with $\mathcal{F}_0$ and the simulator $\mathcal{S}_0^{\mathcal{A}}$ from Figure 3.

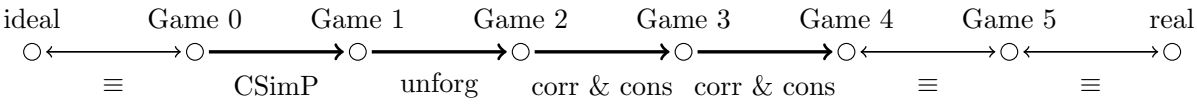

Figure 4: Sequence of Games

Therefore, the probability that $\mathcal{E}$ outputs 1 is the same as in the ideal setting. Hence, the following lemma follows.

Lemma 4.1. *For all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$, it holds that*

$$\Pr\left[\mathcal{E}\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}_0^{\mathcal{A}}\right] = 1\right] = \Pr\left[\mathcal{E}\left[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}\right] = 1\right] \quad .$$

Game 1: (Remove simulation algorithms) In this step we modify $\mathcal{F}_0$ to $\mathcal{F}_1$ and $\mathcal{S}_0^{\mathcal{A}}$ to $\mathcal{S}_1^{\mathcal{A}}$. The goal is to use the algorithms Setup, KeyGen, and Sign instead of SimSetup, SimKeyGen and SimSign of Π_{ABS}.

- In **Setup:** $\mathcal{S}_1^{\mathcal{A}}$ does not send algorithms SimSetup(1^λ), SimKeyGen and SimSign instead it sends Setup(1^λ), KeyGen and Sign to $\mathcal{F}_1$.
- In **Signature:** Step 4 and 3 (Figure 1) are modified. Step 4 is omitted completely. As a result, $\mathcal{F}_1$ never asks $\mathcal{S}_1^{\mathcal{A}}$ for a signature. In step 3 the check ($\mathsf{pp}' = \mathsf{pp}$) is omitted and the rest is changed such that $\mathcal{F}_1$ looks up last recorded entry $(\text{KeyGen}, P_i, \mathsf{pp}', \mathbb{A}, \mathsf{sk}_{\mathbb{A}})$ with $\mathbb{P}(\mathbb{A}) = 1$ and generates the signature as $\text{Sign}(\mathsf{pp}', \mathsf{sk}_{\mathbb{A}}, m, \mathbb{P})$ instead of using SimSign as in $\mathcal{F}_0$.

We change the setting of $\mathcal{E}$ from Game 0 to Game 1, thus from a game with simulated setup, simulated keys and simulated signatures to one with the signatures, keys and setup of π_{ABS}.

Lemma 4.2. *If Π_{ABS} is a computationally simulation private* CSimP *attribute-bases signature scheme, then for all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$, $|\Pr\left[\mathcal{E}\left[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}\right] = 1\right] - \Pr\left[\mathcal{E}\left[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}\right] = 1\right]| \leq \epsilon_{0,1}(\lambda)$, where advantage $\epsilon_{0,1}(\cdot)$ is negligible in λ.*

Proof. We show that if environment $\mathcal{E}$ distinguishes $\left[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}\right]$ and $\left[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}\right]$ with non-negligible probability, then it distinguishes the output of the simulation algorithms from the output of the real algorithms. Let us look at the view of $\mathcal{E}$ in both settings. In $\left[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}\right]$ the instances resulting from the Setup activations are simulated, involving SimSetup, SimKeyGen and SimSign. In $\left[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}\right]$ the instances run the algorithms Setup, KeyGen and Sign. Notice, all algorithms are defined by Π_{ABS}. Given an $\mathcal{E}$ and $\mathcal{A}$ that distinguishes $\left[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}\right]$ from $\left[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}\right]$, we define algorithm D to attack computational simulation privacy CSimP (Definition 2.9).

In the following definition of D, we define how D emulates for $\mathcal{E}$ the behavior of $[\mathcal{F}_b, \mathcal{S}_b^{\mathcal{A}}]$ by interacting with its challenger from the experiment $\text{SP}_D^{\Pi_{\text{ABS}}}(\lambda, b)$. In the definition of D we omit sid and parameter checks, and to shorten the description we leave out the concrete format of the input and output messages, and recording of entries. Further, we refer to Figure 3 to see how corruption works in D. To emulate $[\mathcal{F}_b, \mathcal{S}_b^{\mathcal{A}}]$, D follows the input and output behavior determined by the ideal functionality, i.e. the message formats for inputs and outputs.

Initialization: D on input $(\mathsf{pp}, \mathsf{msk})$ from $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, b)$, D first runs $\mathcal{E}$ and $\mathcal{A}$. If $\mathcal{E}$ and $\mathcal{A}$ want to exchange messages, i.e. activations and results of a corruptions, then D lets them communicate. Overall, whenever $\mathcal{F}_{\mathrm{ABS}}$ would use the algorithms S.KG or S.Sign provided by a simulator, D will relate it to $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, b)$ and form responses using the provided oracles.

Setup: On $\mathcal{E}$ sending $(\mathrm{Setup}, \mathrm{sid})$ to P_{Setup}, D outputs $(\mathrm{Public\,Params}, \mathrm{sid}, \mathsf{pp})$ where pp is given by the experiment

Key Generation: On $\mathcal{E}$ sending $(\mathrm{KeyGenRequest}, \mathrm{sid}, \mathrm{kid}, \mathbb{A}_l)$ to a party P.

- D behaves like $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ (not changed in $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$)

Key Generation: On $\mathcal{E}$ sending $(\mathrm{KeyGen}, \mathrm{sid}, \mathrm{kid})$ to party P_{Setup}.

- If P and P_{Setup} are honest, D behaves like $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ but for key generations it queries oracle $\mathcal{O}_{\mathsf{pp},\mathsf{msk}}^{\mathrm{KeyGen}_b}(\mathbb{A}_l)$.
- If P is corrupt and P_{Setup} is honest, D first queries $\mathcal{O}_{\mathsf{pp},\mathsf{msk}}^{\mathrm{KeyGen}_b}(\mathbb{A}_l)$ and then queries $\mathcal{O}^{\mathrm{Reveal}}(l)$ to get $\mathsf{sk}_{\mathbb{A}_l}$, and outputs $(\mathrm{KeyGen}, \mathrm{sid}, \mathrm{kid}, \mathsf{pp}, \mathbb{A}_l, \mathsf{sk}_{\mathbb{A}_l})$ to $\mathcal{A}$ for P.
- If P_i is honest and P_{Setup} is corrupt, D follows the steps of $\mathcal{S}_0^{\mathcal{A}}$ in Figure 3 (not changed in $\mathcal{S}_1^{\mathcal{A}}$) to simulate the communication with the corrupted party. This includes the recording of $(\mathrm{KeyGen}, \mathrm{sid}, \mathrm{kid}, \mathsf{pp}', \mathbb{A}_l, P_i, \mathsf{sk}_{\mathbb{A}_l})$.

Signature: If $\mathcal{E}$ sends $(\mathrm{Signature}, \mathrm{sid}, \mathsf{pp}', \mathbb{A}_i, m_j, \mathbb{P}_j)$ for a corrupted party P_l, D lets $\mathcal{A}$ handle it as the simulator $\mathcal{S}_b^{\mathcal{A}}$ would do. If P_l is honest and

- $\mathsf{pp}' = \mathsf{pp}$: D answers with a signature from $\mathcal{O}_{\mathsf{pp},\mathsf{msk}}^{\mathrm{Sign}_b}(i, m_j, \mathbb{P}_j)$
- $\mathsf{pp}' \neq \mathsf{pp}$: If $(P_l, \mathsf{pp}', \mathbb{A}_i, \mathsf{sk}_{\mathbb{A}_i})$ is recorded, D answers with a signature by running $\mathrm{Sign}(\mathsf{pp}', \mathsf{sk}_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$.

Verification: If $\mathcal{E}$ sends $(\mathrm{Verify}, \mathrm{sid}, \mathsf{pp}, m, \mathbb{P}, \sigma)$ to a corrupted P_l, D lets $\mathcal{A}$ handle it. If P_l is honest, then D executes the verification steps of $\mathcal{F}_0$ (same as in $\mathcal{F}_1$).

Output: If $\mathcal{E}$ outputs a bit $\tilde{b}$, D outputs it as well.

Analysis: First of all, the view of $\mathcal{E}$ regarding the verification was not altered at all, since the Verify activation was not changed from $\mathcal{F}_0$ to $\mathcal{F}_1$. In the case of $\mathsf{pp}' \neq \mathsf{pp}$ in a Signature activations, $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ and $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$ use Sign to generate a signature under pp'. The same holds for D. All the other steps of D also perfectly emulate the behavior of $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ and $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$.

If D is taking part in $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 0)$, then the public parameters given by the game are output by SimSetup, keys returned by $\mathcal{O}_{\mathsf{pp},\mathsf{msk}}^{\mathrm{KeyGen}_0}$ are generated by SimKeyGen, and signatures returned by the $\mathcal{O}_{\mathsf{pp},\mathsf{msk}}^{\mathrm{Sign}_0}$ are computed using SimSign. Therefore, the view of $\mathcal{E}$ in this case is distributed as in $\left[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}\right]$. Suppose that D is taking part in $\mathrm{SP}_D^{\Pi_{\mathrm{ABS}}}(\lambda, 1)$, then the given public parameters are the output of Setup, keys returned by $\mathcal{O}_{\mathsf{pp},\mathsf{msk}}^{\mathrm{KeyGen}_1}$ are generated by

KeyGen, and the signatures returned by $\mathcal{O}^{\text{Sign}_1}_{\mathsf{pp},\mathsf{msk}}$ are computed using the Sign algorithm. Hence the view of $\mathcal{E}$ is distributed as in $\left[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}\right]$. At the end the final output of D is the output of $\mathcal{E}$ (0 or 1). Overall, for a computationally simulation private ABS scheme Π_{ABS} we can conclude that, $\left|\Pr\left[\mathcal{E}\left[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}\right] = 1\right] - \Pr\left[\mathcal{E}\left[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}\right] = 1\right]\right| \leq |\Pr[\text{SP}_D^{\Pi_{\text{ABS}}}(\lambda, 0) = 1] - \Pr[\text{SP}_D^{\Pi_{\text{ABS}}}(\lambda, 1) = 1]| = \text{Adv}_D^{\text{CSimP}}(\lambda)$, where the advantage $\text{Adv}_D^{\text{CSimP}}(\lambda)$ is negligible in λ.

□

Game 2: (Remove absolute unforgeability) With absolute unforgeability we refer to the property that for honest parties under registered pp our ideal ABS functionality guarantees unforgability with probability 1. In this step we modify $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$ and denote the result as $[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}]$, where we only modify $\mathcal{F}_1$ and the simulator is not changed, $\mathcal{S}_1^{\mathcal{A}} = \mathcal{S}_2^{\mathcal{A}}$. To define $\mathcal{F}_2$, we introduce one modification to the Verify step I.iii. (see Figure 1, the step was not changed up to now), where $f_{\text{out}} := 0$ is set. This step is changed to $f_{\text{out}} := b$, where b is the output of the verification done with S.Verify.

Lemma 4.3. *If Π_{ABS} is unforgeable (Definition 2.5), then for all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$, $|\Pr\left[\mathcal{E}\left[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}\right] = 1\right] - \Pr[\mathcal{E}\left[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}\right] = 1]| \leq \epsilon_{1,2}(\lambda)$, where advantage $\epsilon_{1,2}(\cdot)$ is negligible in λ.*

Proof. Notice that the modification of Game 2 introduces a difference to Game 1 in Verify step I.iii only if $b = 1$, i.e. $f = 1$, holds at this step. We will show that this event only appears if the unforgeability of the scheme was broken. Let us denote the event that $b = 1$ holds in step I.iii. with $\text{Event}_{\text{Forge}}$. First, let us make clear what it means if Verify step I.iii. of $\mathcal{F}_1$ (same step as in Figure 1) is reached and $\text{Event}_{\text{Forge}}$ happens for input $(\text{Verify}, \text{sid}, \mathsf{pp}, m, \mathbb{P}, \sigma)$. If the verification of $\mathcal{F}_1$ and $\mathcal{F}_2$ reaches step I.iii. we know that the conditions for the previous steps did not hold. In particular, we know that the signature on $(m, \mathbb{P})$ was not generated by the ideal functionality and it was not verified in a previous Verify activation (step I.i.). Further, we know that $(m, \mathbb{P})$ was never signed (step I.ii.). Importantly, from step I.ii. it follows that P_{Setup} is honest. Hence, as in $\text{Exp}_F^{\text{UF}}(\lambda)$ the master secret is kept secret. Furthermore, from step I.ii. it follows that there is no corrupted signer P_i with a record $(\text{KeyGen}, \text{sid}, \text{kid}, P_i, \mathsf{pp}, \mathbb{A}', \cdot)$ where $\mathbb{P}(\mathbb{A}') = 1$. Therefore, no corrupted party could have legitimately generated the signature. We can conclude that if $\text{Event}_{\text{Forge}}$ happens, then the signature is one of a party without a satisfying secret key for $\mathbb{P}$ and the signature was not generated by $\mathcal{F}_{\text{ABS}}$, i.e. it is a forgery.

Let us define the forger F for an adversary $\mathcal{A}$ and environment $\mathcal{E}$. Similar to the distinguisher from Lemma 4.2, forger F behaves like the ideal functionality and simulator $[\mathcal{F}_1], \mathcal{S}_1^{\mathcal{A}}$. We just describe the situations where F behaves differently to trigger the event. Overall, F follows the input and output message format as defined by $\mathcal{F}_{\text{ABS}}$ in Figure 1 (same as in $\mathcal{F}_1$ and $\mathcal{F}_2$). Also, F uses the oracle access provided by the unforgeability experiment $\text{Exp}_F^{\text{UF}}(\lambda)$ to answer $\mathcal{E}$'s Sign and Key Generation activations, instead of using the algorithms provided by the simulator. To shorten the description we omit the output messages, sid and parameter checks, and details of the corruption. They are the same as in Figure 1 and 3.

Initialization: On input pp from $\mathrm{Exp}_F^{\mathrm{UF}}(\lambda)$, F runs $\mathcal{E}$ and $\mathcal{A}$. Messages between $\mathcal{A}$ and $\mathcal{E}$ are forwarded by F. F emulates $\mathcal{F}_{\mathrm{ABS}}$ and $\mathcal{S}$ as described in the following.

Setup: If $\mathcal{E}$ sends $(\mathrm{Setup}, \mathrm{sid})$ to P_{Setup}, F returns $(\mathrm{Public\,Params}, \mathrm{sid}, \mathsf{pp})$

Key Generation $\mathbb{A}_i$: F while simulating the key generation part of P_{Setup}. For honest party P_i, F queries the oracle $\mathcal{O}^{\mathrm{KeyGen}}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_i)$ and returns $(\mathrm{KeyGen}, \mathrm{sid}, \mathrm{kid}, \mathbb{A}_i)$. If P_i is corrupt, F queries $\mathcal{O}^{\mathrm{KeyGen}}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_i)$ gets $\mathsf{sk}_{\mathbb{A}_i}$ by querying $\mathcal{O}^{\mathrm{Reveal}}(i)$ and outputs $(\mathrm{KeyGen}, \mathrm{sid}, \mathrm{kid}, \mathsf{pp}, \mathbb{A}_i, \mathsf{sk}_{\mathbb{A}_i})$ to $\mathcal{A}$ for P_i.

Signature: If $\mathcal{E}$ sends $(\mathrm{Signature}, \mathrm{sid}, \mathsf{pp}, \mathbb{A}_i, m_j, \mathbb{P}_j)$ to a corrupted party P_l, F lets $\mathcal{A}$ handle it. If P_l is honest and a key for $(\mathbb{A}_i, P_l)$ with $\mathbb{P}_j(\mathbb{A}_i) = 1$ was generated, F computes the signature with a query to $\mathcal{O}^{\mathrm{Sign}_b}_{\mathsf{pp},\mathsf{msk}}(i, m_j, \mathbb{P}_j)$, otherwise F ignores the activation.

Verification/Output: If $\mathcal{E}$ sends $(\mathrm{Verify},\ \mathrm{sid}, \mathsf{pp}', m, \mathbb{P}, \sigma)$ to a corrupted party P_l, F lets $\mathcal{A}$ handle it. If P_l is honest, F executes the verification steps of $\mathcal{F}_{\mathrm{ABS}}$. If $\mathrm{Event}_{\mathrm{Forge}}$ happens during the verification checks, F outputs $(m, \mathbb{P}, \sigma)$ as its forgery.

F answers the activation send by $\mathcal{E}$ with cryptographic elements generated by the oracles. The oracles use the same algorithms $(\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ as the simulator $\mathcal{S}_2^{\mathcal{A}}$ of the current Game 2. From the argumentation above it follows that if $\mathcal{E}$ causes $\mathrm{Event}_{\mathrm{Forge}}$ forger F outputs a forgery and wins $\mathrm{Exp}_F^{\mathrm{UF}}(\lambda)$. □

Game 3: (Verification with algorithm Verify) To define $[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}]$ we modify $[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}]$ such that the interaction between them works as follows.

$\mathcal{F}_3$ on input $(\mathrm{Verify}, \mathrm{sid}, \mathsf{pp}', m, \mathbb{P}, \sigma)$ from some party P

1. Send $(\mathrm{Verify}, \mathrm{sid}, \mathsf{pp}', m, \mathbb{P}, \sigma)$ to $\mathcal{S}_3^{\mathcal{A}}$ and $\mathcal{S}_3^{\mathcal{A}}$ runs $b \leftarrow \mathrm{Verify}(\mathsf{pp}', m, \mathbb{P}, \sigma)$ and sends $(\mathrm{Verified}, \mathrm{sid}, \mathsf{pp}', m, \mathbb{P}, \sigma, b)$ to $\mathcal{F}_3$.
2. Upon $(\mathrm{Verified}, \mathrm{sid}, \mathsf{pp}', m, \mathbb{P}, \sigma, b)$ from $\mathcal{S}_3^{\mathcal{A}}$ record $(\mathrm{Signature}, \mathrm{sid},\ \mathsf{pp}', m,\ \mathbb{P}, \sigma, b)$ and output $(\mathrm{Verified}, \mathrm{sid}, m, \mathbb{P}, \sigma, b)$ to P.

This removes the verification checks that were present in $\mathcal{F}_2$.

Lemma 4.4. *For ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$ it holds that*

$$|\Pr\left[\mathcal{E}\left[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}\right] = 1\right] - \Pr\left[\mathcal{E}\left[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}\right] = 1\right]| \leq \epsilon_{2,3}(\lambda)\ ,$$

where $\epsilon_{2,3}(\cdot)$ is negligible in λ.

Proof. We have to show that for every possible verification input to $\mathcal{F}_2$ the verification result already was equal to the output that algorithm Verify, used in $\mathcal{F}_3$, would have produced. First, we have to observe that correctness and consistency may fail with negligible probability. Therefore, we show that every step in the verification of $\mathcal{F}_2$ followed the output bit b of Verify. The steps I.ii., I.iii., and II.ii. in $\mathcal{F}_2$ (see Figure 1) already set $f_{\mathrm{out}} = b$, for b the output of Verify.

The critical steps are I.i. and II.i. In both steps the bit f_{out} is set to a previously recorded bit f, but due to the changes in Game 3 we now use the bit b of Verify in $\mathcal{F}_3$. The bit f could have been previously recorded in two cases. First during the corresponding Signature activation (step 6 of Signature in Figure 1) and second in a Verification activation. In the first case, for honest signers, it follows from the correctness that $f \neq b$ only occurs with negligible probability. For corrupted signers the same follows from the consistency. In the second case, we can conclude that f was recorded in step I.ii., I.iii. or II.ii., thus the bit b was already used in $\mathcal{F}_2$ of Game 2. Overall, the environment $\mathcal{E}$ can distinguish only if the correctness or consistency fails. □

Game 4: (Remove halting condition in Signature activation) We use $\left[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}\right]$ as a starting point and modify them to define $\left[\mathcal{F}_4, \mathcal{S}_4^{\mathcal{A}}\right]$. We remove from $\mathcal{F}_3$ the halting condition in step 5 of the Signature activation (was still unchanged from $\mathcal{F}_0$ in Figure 1) and directly record $(\text{Signature}, \text{sid}, \mathsf{pp}, m, \mathbb{P}, \sigma, 1)$ and output $(\text{Signature}, \text{sid}, \mathbb{A}, m, \mathbb{P}, \sigma)$ to P_i in step 6. The simulator is unchanged ($\mathcal{S}_4^{\mathcal{A}} = \mathcal{S}_3^{\mathcal{A}}$).

Lemma 4.5. *For all ppt adversaries $\mathcal{A}$ and all environments $\mathcal{E}$ it holds that*

$$|\Pr\left[\mathcal{E}\left[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}\right] = 1\right] - \Pr\left[\mathcal{E}\left[\mathcal{F}_4, \mathcal{S}_4^{\mathcal{A}}\right] = 1\right]| \leq \epsilon_{3,4}(\lambda) \ ,$$

where $\epsilon_{3,4}(\cdot)$ is negligible in λ.

Proof. In Signature step 5 (Figure 1) the condition that a record $(\text{Signature}, \text{sid}, \mathsf{pp}, m, \mathbb{P}, \sigma, 0)$ exists can only hold if previously a Verification activation $(\text{Verify}, \text{sid}, \mathsf{pp}, m, \mathbb{P}, \sigma)$ recorded $f_{\text{out}} = 0$. However, this would mean that the output of $\text{Verify}(\mathsf{pp}, m, \mathbb{P}, \sigma)$ was $b = 0$. Since correctness holds, we can conclude that signatures for honest signers generated by the algorithm Sign are verified by algorithm Verify with $b = 0$ only with negligible probability. From the consistency we can conclude the following. If the same signature σ was previously generated by a corrupted signer and verified by a Verification activation (in other words by Verify) the result $f_{\text{out}} = 0$ would have been recorded only with negligible probability. □

Game 5: (Remove records for signatures) Game 5 is a modification of Game 4 where $\mathcal{S}_5^{\mathcal{A}} = \mathcal{S}_4^{\mathcal{A}}$. Only $\mathcal{F}_4$ is modified to $\mathcal{F}_5$, such that it does not create records for signatures like $(\text{Signature}, \text{sid}, \mathsf{pp}', m', \mathbb{P}', \sigma', b')$ in Signature and Verification activations.

Lemma 4.6. *For all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$ it holds that*

$$\Pr\left[\mathcal{E}\left[\mathcal{F}_4, \mathcal{S}_4^{\mathcal{A}}\right] = 1\right] = \Pr\left[\mathcal{E}\left[\mathcal{F}_5, \mathcal{S}_5^{\mathcal{A}}\right] = 1\right] \ .$$

Proof. $\mathcal{F}_5$ performs only parameter and sid checks, asks $\mathcal{S}_5^{\mathcal{A}}$ and returns what $\mathcal{S}_5^{\mathcal{A}}$ and the algorithms produced as output. The same algorithms as in π_{ABS} are used. Through the modifications in Game 4 the records of the signatures were never read by $\mathcal{F}_4$. Hence, the view for a $\mathcal{E}$ is not changed; we just removed unused records. □

At this point $\mathcal{F}_5$ in comparison to $\mathcal{F}_0$ is just an parameter checker and every behavior and output is determined by the simulator. The simulator $\mathcal{S}_0^{\mathcal{A}}$ was modified such that is resembles π_{ABS} in $\mathcal{S}_5^{\mathcal{A}}$.

Lemma 4.7. *For all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$ it holds that*

$$\Pr\left[\mathcal{E}\left[\mathcal{F}_5, \mathcal{S}_5^{\mathcal{A}}\right] = 1\right] = \Pr\left[\mathcal{E}\left[\pi_{\mathrm{ABS}}, \mathcal{A}\right] = 1\right] \ .$$

Proof. For every activation the output of $\mathcal{F}_5$ is described by the output of $\mathcal{S}_5^{\mathcal{A}}$ and the algorithms $(\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ used in the simulator, where $\mathcal{S}_5^{\mathcal{A}}$ simulates the communication of the honest parties with $\mathcal{A}$. Overall, the same algorithms with the same inputs are used as in π_{ABS}. Further, as in π_{ABS} the simulator $\mathcal{S}_5^{\mathcal{A}}$ let $\mathcal{A}$ determine the behavior and output of corrupted parties. Consequently, the view of $\mathcal{E}$ in $\left[\mathcal{F}_5, \mathcal{S}_5^{\mathcal{A}}\right]$ is equal to the view of $\mathcal{E}$ in $[\pi_{\mathrm{ABS}}, \mathcal{A}]$. □

Proof of Theorem 4.2: From Lemmas 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, and 4.7 it follows

$$\begin{aligned} &\left|\Pr\left[\mathcal{E}\left[\mathcal{F}_{\mathrm{ABS}}, \mathcal{S}_0^{\mathcal{A}}\right] = 1\right] - \Pr\left[\mathcal{E}\left[\pi_{\mathrm{ABS}}, \mathcal{A}\right](\lambda) = 1\right]\right| \\ &\qquad \leq \epsilon_{0,1}(\lambda) + \epsilon_{1,2}(\lambda) + \epsilon_{2,3}(\lambda) + \epsilon_{3,4}(\lambda) \\ &\qquad \leq \nu(\lambda) \end{aligned}$$

where $\nu(\cdot)$ is negligible. Hence, the protocol π_{ABS} realizes the ideal ABS functionality $\mathcal{F}_{\mathrm{ABS}}$[adaptive*, erasure, secure channels].

4.2 UC Security implies Experiment-Based Security

We show the second part of Theorem 4.1 in the following.

Theorem 4.3. *If π_{ABS} realizes $\mathcal{F}_{\mathrm{ABS}}$ [adaptive*, erasure, secure channels], then Π_{ABS} is correct, consistent, unforgeable and computationally simulation private.*

The theorem follows from the lemmas 4.8, 4.9, 4.10 and 4.11 stated below. Regarding the lemmas we use the contraposition of Theorem 4.3; Π_{ABS} is **not** (correct $\wedge$ consistent $\wedge$ unforgeable $\wedge$ computationally simulation private) implies that π_{ABS} does not realize $\mathcal{F}_{\mathrm{ABS}}$. Where π_{ABS} does not realize $\mathcal{F}_{\mathrm{ABS}}$ denotes, that there is an adversary $\mathcal{A}$ such that for all simulators $\mathcal{S}$, there exists an environment $\mathcal{E}$ such that $\mathcal{E}\left[\mathcal{F}_{\mathrm{ABS}}, \mathcal{S}\right]$ and $\mathcal{E}\left[\pi_{\mathrm{ABS}}, \mathcal{A}\right]$ are not computationally indistinguishable in respect to the security parameter.

Lemma 4.8. *Assume the ABS scheme Π_{ABS} is not correct, then π_{ABS} does not realize the ideal functionality $\mathcal{F}_{\mathrm{ABS}}$ [adaptive*, erasure, secure channels].*

Proof. If Π_{ABS} is not correct (Definition 2.2) then there exists a message m, attribute set $\mathbb{A}$ and policy $\mathbb{P}$ such that, $\Pr\left[(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathrm{Setup}\left(1^{\lambda}\right); \mathsf{sk}_{\mathbb{A}} \leftarrow \mathrm{KeyGen}\left(\mathsf{pp}, \mathsf{msk}, \mathbb{A}\right); \sigma \leftarrow \mathrm{Sign}\left(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}}, m, \mathbb{P}\right) : \mathrm{Verify}\left(\mathsf{pp}, m, \mathbb{P}, \sigma\right) = 0\right] = \tau(\lambda)$ is non-negligible. The environment $\mathcal{E}$ works as follows in this case. $\mathcal{E}$ first sets $\mathrm{sid} = (P_{\mathrm{Setup}}, 0)$ and runs the setup through P_{Setup} with input $(\mathrm{Setup}, \mathrm{sid})$ and obtains as a result the public parameter pp. Secondly, $\mathcal{E}$ activates some party P with $(\mathrm{KeyGenRequest}, \mathrm{sid}, \mathrm{kid}, \mathbb{A})$ and then sends to corresponding activation $(\mathrm{KeyGen}, \mathrm{sid}, \mathrm{kid})$ to P_{Setup}. After that it activates the same party P with $(\mathrm{Signature}, \mathrm{sid}, \mathsf{pp}, \mathbb{A}, m, \mathbb{P})$ and obtains the signature σ. At last $\mathcal{E}$ activates a party P_V with $(\mathrm{Verify}, \mathrm{sid}, \mathsf{pp}, m, \mathbb{P}, \sigma)$ to verify

the signature and outputs the result. Observe that $\mathcal{E}$ will always output 1 in the ideal setting $[\mathcal{F}_{\text{ABS}}, \mathcal{S}]$, but outputs 0 with non-negligible probability in the real setting $[\pi_{\text{ABS}}, \mathcal{A}]$. Hence, $\left| \Pr\left[\mathcal{E}\left[\pi_{\text{ABS}}, \mathcal{A}\right](\lambda) = 1\right] - \Pr\left[\mathcal{E}\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}\right](\lambda) = 1\right] \right| = \tau(\lambda)$. □

Lemma 4.9. *Assume* Π_{ABS} *is not consistent, then* π_{ABS} *does not realize the ideal functionality* $\mathcal{F}_{\text{ABS}}$ [*adaptive*, erasure, secure channels*].

Proof. To make it short, it is a similar argument as in the proof of Lemma 4.8 and $\mathcal{E}$ works the same way as defined there, except that it activates P_V twice with $(\text{Verify}, \text{sid}, \mathsf{pp}, m, \mathbb{P}, \sigma)$. $\mathcal{E}$ outputs 1 if and only if the two returned verification values are equal. If we analyze the ideal setting $[\mathcal{F}_{\text{ABS}}, \mathcal{S}]$, then $\mathcal{E}$ always outputs 1, because the ideal verification guarantees absolute consistency. In the real setting $[\pi_{\text{ABS}}, \mathcal{A}]$ the environment $\mathcal{E}$ outputs 0 with non-negligible probability, since Π_{ABS} used in π_{ABS} is not consistent. □

4.2.1 Privacy

To show simulation privacy we define an environment that distinguishes the output of the simulation algorithms and the non-simulation algorithms.

Lemma 4.10. *Assume* Π_{ABS} *is correct and consistent, but not computationally simulation private* CSimP*, then* π_{ABS} *does not realize* $\mathcal{F}_{\text{ABS}}$[*adaptive*, erasure, secure channels*].

Let us first outline the proof. Since we assume that Π_{ABS} is not CSimP, it holds that for all tuples of ppt algorithms (SimSetup, SimKeyGen, SimSign) according to Definition 2.9, there exists a distinguisher D that distinguishes the experiment $\text{SP}_D^{\Pi_{\text{ABS}}}(\lambda, 0)$ from $\text{SP}_D^{\Pi_{\text{ABS}}}(\lambda, 1)$ with non-negligible advantage.

For a fixed adversary $\mathcal{A}$ and for every simulator $S^{\mathcal{A}}$ we show how to define such a tuple of algorithms (SimSetup, SimKeyGen, SimSign). Since Π_{ABS} is not CSimP, there exists a distinguisher $D_{\mathcal{S}^{\mathcal{A}}}$ for these three algorithms (SimSetup, SimKeyGen, SimSign). We use this distinguisher $D_{\mathcal{S}^{\mathcal{A}}}$ to define an environment $\mathcal{E}$ that distinguishes the two settings $\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}^{\mathcal{A}}\right]$ and $[\pi_{\text{ABS}}, \mathcal{A}]$.

Proof. In the following we use a specialized adversary $\mathcal{A}$ that corrupts the setup party P_{Setup} after the Setup activation. That means if it sees the output (Public Params, sid, pp) from P_{Setup} it corrupts P_{Setup} and outputs $(\mathsf{pp}, \mathsf{msk})$ to the environment. After that it lets the environment handle P_{Setup} and therefore forwards every message for P_{Setup} to the environment. Additionally, $\mathcal{A}$ corrupts every party P_i after a completed Key Generation activation, i.e. if $\mathcal{A}$ sees public delayed output $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A})$. For every other message it behaves as the standard adversary.

In the following we set $\text{sid} = (P_{\text{Setup}}, 0)$. As defined through $\mathcal{F}_{\text{ABS}}$ the simulator $S^{\mathcal{A}}$ has to send, among other things, stateless ppt algorithms (S.Setup, S.KG, S.Sign) to $\mathcal{F}_{\text{ABS}}$. For the following algorithms fix randomness r.

SimSetup $\left(1^{\lambda}\right)$**:** On input security parameter 1^{λ}, SimSetup starts $\mathcal{S}^{\mathcal{A}}$ with randomness r and sends $(\text{Setup}, \text{sid})$ to $\mathcal{S}^{\mathcal{A}}$. Upon receiving (Setup, sid, S.Setup, S.KG, S.Sign, S.Verify) from $\mathcal{S}^{\mathcal{A}}$, SimSetup generates $(\mathsf{pp}, \mathsf{msk})$ with S.Setup and outputs (Public Params, sid, pp). Subsequently $\mathcal{S}^{\mathcal{A}}$ corrupts P_{Setup} and $\mathcal{S}^{\mathcal{A}}$ outputs $(\mathsf{pp}, \mathsf{msk})$. SimSetup outputs $(\mathsf{pp}, \mathsf{msk})$.

SimKeyGen $(\mathsf{pp}, \mathsf{msk}, \mathbb{A}_i)$**:** The algorithm runs the steps of SimSetup up to the point where P_{Setup} gets corrupted. Next, SimKeyGen executes the steps of the Key Generation activation $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A}_i)$ for a honest party P_i and an unique kid as in $[\mathcal{F}_{\text{ABS}}, \mathcal{S}_0^{\mathcal{A}}]$. Therefore, it delegates public delayed output to $\mathcal{S}^{\mathcal{A}}$ including the request $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}, P_i)$. Upon public delayed output $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A}_i)$, $\mathcal{S}^{\mathcal{A}}$ corrupts P_i. SimKeyGen gets $\mathsf{sk}_{\mathbb{A}_i}$ from the output of the corruption and outputs $\mathsf{sk}_{\mathbb{A}_i}$.

SimSign $(\mathsf{pp}, \mathsf{msk}, m, \mathbb{P})$**:** The algorithm runs the steps of SimSetup up to the point where P_{Setup} gets corrupted. Then SimSign uses the Signature activation of $\mathcal{F}_{\text{ABS}}$ to get the signature on $(m, \mathbb{P})$. Eventually the output $(\text{Signature}, \text{sid}, \mathbb{A}, m, \mathbb{P}, \sigma)$ is created and SimSign outputs σ.

Notice that SimSetup, SimKeyGen and SimSign are getting the same algorithms from $\mathcal{S}^{\mathcal{A}}$, since they use the same fixed randomness r to run $\mathcal{S}^{\mathcal{A}}$.

From the assumption we know Π_{ABS} is not CSimP. Hence for the algorithms (SimSetup, SimKeyGen, SimSign) as defined above, there exists a distinguisher $D_{\mathcal{S}^{\mathcal{A}}}$ that distinguishes $\text{SP}^{\Pi_{\text{ABS}}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda, b = 0)$ and $\text{SP}^{\Pi_{\text{ABS}}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda, b = 1)$ with non-negligible advantage $\text{Adv}^{\text{CSimP}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda)$. Next, we define the environment $\mathcal{E}$, for $\mathcal{A}$ and arbitrary but fixed $\mathcal{S}^{\mathcal{A}}$. Note, $\mathcal{E}$ takes the role of P_{Setup} as $\mathcal{A}$ will forward every message for P_{Setup} after the corruption of P_{Setup}. Important is that $\mathcal{E}$ will act as an honest P_{Setup} and answers to requests as described in $\mathcal{F}_{\text{ABS}}$. The setup party is corrupted only to get the master secret.

1. $\mathcal{E}$ sends $(\text{Setup}, \text{sid})$ to P_{Setup} and gets $(\text{Public Params}, \text{sid}, \mathsf{pp})$ back. From the corruption of P_{Setup} it gets $(\mathsf{pp}, \mathsf{msk})$.

2. $\mathcal{E}$ starts $D_{\mathcal{S}^{\mathcal{A}}}$ on input $(\mathsf{pp}, \mathsf{msk})$ and answers $D_{\mathcal{S}^{\mathcal{A}}}$'s oracle queries as follows.

   $\mathcal{O}^{\text{KeyGen}_b}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_i)$: On i-th query input $\mathbb{A}_i$, environment $\mathcal{E}$ activates a new party P_i with $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A}_i)$ and takes the role of corrupted P_{Setup}. Therefore, $\mathcal{E}$ eventually gets message $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A}_i)$ for P_{Setup} from P_i forwarded by $\mathcal{A}$. $\mathcal{E}$ generates $\mathsf{sk}_{\mathbb{A}_i} \leftarrow \text{KeyGen}(\mathsf{pp}, \mathsf{msk}, \mathbb{A}_i)$, and sends message $(\text{KeyGen}, \text{sid}, \text{kid}, \mathsf{pp}, \mathbb{A}_i, \mathsf{sk}_{\mathbb{A}_i})$ to P_i. After that $\mathcal{E}$ outputs $\mathsf{sk}_{\mathbb{A}_i}$ to $D_{\mathcal{S}^{\mathcal{A}}}$.

   $\mathcal{O}^{\text{Sign}_b}_{\mathsf{pp},\mathsf{msk}}(i, m_j, \mathbb{P}_j)$: On j-th query, environment $\mathcal{E}$ checks if $\mathbb{A}_i$ was queried, that $\mathbb{P}_j(\mathbb{A}_i) = 1$ holds and that $\mathbb{P}_j \in \mathbb{U}(\mathsf{pp})$. If not, it ignores the query. Otherwise, $\mathcal{E}$ activates a party P_i with $(\text{Signature}, \text{sid}, \mathsf{pp}, \mathbb{A}_i, m_j, \mathbb{P}_j,)$, eventually party P_i returns $(\text{Signature}, \text{sid}, \mathbb{A}_i, m_j, \mathbb{P}_j, \sigma_j)$ and $\mathcal{E}$ outputs σ_j to $D_{\mathcal{S}^{\mathcal{A}}}$.

3. Eventually $D_{\mathcal{S}^{\mathcal{A}}}$ outputs $\tilde{b}$ and $\mathcal{E}$ outputs it.

Analysis: We will now relate the advantage of $\mathcal{E}$ to the advantage of $D_{\mathcal{S}^{\mathcal{A}}}$ in the computational simulation private experiment $\text{SP}^{\Pi_{\text{ABS}}}_{D_{\mathcal{S}^{\mathcal{A}}}}$. We analyze the real and the ideal settings separately.

Case $[\pi_{\text{ABS}}, \mathcal{A}]$**:** From the definition of π_{ABS} (Figure 2) we get the following. The tuple $(\mathsf{pp}, \mathsf{msk})$ is generated by $\text{Setup}(1^\lambda)$ and the secret keys $\mathsf{sk}_{\mathbb{A}_i}$ are generated by the algorithm KeyGen with input pp, msk and attribute set $\mathbb{A}_i$. The signatures σ_j are generated by an execution of $\text{Sign}(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$. Thus the view of $D_{\mathcal{S}^{\mathcal{A}}}$ is equal to the view in $\text{SP}^{\Pi_{\text{ABS}}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda, 1)$, with algorithms Setup, KeyGen, Sign, and Verify as defined by Π_{ABS}, which is used in π_{ABS}.

Case $\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}^{\mathcal{A}}\right]$**:** In this case the tuple $(\mathsf{pp}, \mathsf{msk})$ is generated by the algorithm S.Setup output by $\mathcal{S}^{\mathcal{A}}$. The same algorithm is used in our SimSetup above. Further, the Signature activations from $\mathcal{E}$ result in signatures generated by S.Sign in $\mathcal{F}_{\text{ABS}}$. This is the same process as specified by SimSign above. The Key Generation activations are answered by $\mathcal{E}$, with the use of KeyGen. In the setting $[\mathcal{F}_{\text{ABS}}, \mathcal{S}^{\mathcal{A}}]$ with a corrupted P_{Setup} the secret keys are also generated by $\mathcal{E}$, due to the definition of our adversary $\mathcal{A}$ that hands the responsibility for P_{Setup} to $\mathcal{E}$. Hence, the key generation is done as specified by SimKeyGen. Consequently, the view of $D_{\mathcal{S}^{\mathcal{A}}}$ is equal to the view in $\text{SP}^{\Pi_{\text{ABS}}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda, 0)$, for the above defined ppt algorithms SimSetup, SimKeyGen, and SimSign.

From the analysis it follows that,

$$\begin{aligned}
&\left| \Pr\left[\mathcal{E}\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}^{\mathcal{A}}\right] = 1\right] - \Pr\left[\mathcal{E}\left[\pi_{\text{ABS}}, \mathcal{A}\right] = 1\right] \right| \\
=&\left| \Pr\left[\text{SP}^{\Pi_{\text{ABS}}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda, 0) = 1\right] - \Pr\left[\text{SP}^{\Pi_{\text{ABS}}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda, 1) = 1\right] \right| \\
=&\text{Adv}^{\text{CSimP}}_{D_{\mathcal{S}^{\mathcal{A}}}}(\lambda) \quad .
\end{aligned}$$

□

4.2.2 Unforgeability

For showing unforgeability we will construct an environment $\mathcal{E}$ that uses a forger to distinguish the real and the ideal setting.

Lemma 4.11. *Assume* Π_{ABS} *is correct, consistent, computationally simulation private, but not unforgeable, then* π_{ABS} *does not realize* $\mathcal{F}_{\text{ABS}}$*[adaptive*, erasure, secure channels].*

Proof. If Π_{ABS} is not unforgeable as defined in Definition 2.5 but correct, consistent and CSimP, there is a ppt forger F with non-negligible success probability $\tau(\lambda)$ in $\text{Exp}^{\text{UF}}_{F}(\lambda)$. We construct environment $\mathcal{E}$ and $\mathcal{A}$ in the following. Consider $\mathcal{A}$ as the standard adversary. Note, if $\mathcal{E}$ wants to corrupts a party, $\mathcal{A}$ outputs the list of the secret keys of the corrupted party to $\mathcal{E}$. Hence, every simulator S has to do the same for corrupted parties. If not, then the settings are easy to distinguish.

$\mathcal{E}$ first runs the setup through some party P_{Setup} with $(\text{Setup}, \text{sid})$, where $\text{sid} := (P_{\text{Setup}}, 0)$, and obtains as a result (passed on by P_{Setup}) the public parameters pp. Environment $\mathcal{E}$ runs a copy of forger F with pp. Let us describe the interaction of $\mathcal{E}$ with F, which involves the description of how $\mathcal{E}$ answers F's oracle queries. Initially counters i and j are 0.

$\mathcal{O}^{\text{KeyGen}}_{\mathsf{pp},\mathsf{msk}}(\mathbb{A}_i)$: On i-th query input $\mathbb{A}_i$, environment $\mathcal{E}$ activates a party P_i with a key generation $(\text{KeyGenRequest}, \text{sid}, \text{kid}, \mathbb{A}_i)$. Eventually $\mathcal{E}$ gets $(\text{KeyGen}, \text{sid}, \text{kid}, \mathbb{A}, P_i)$ back for a finished key generation by P_{Setup} and stores it. $\mathcal{E}$ increments i.

$\mathcal{O}^{\text{Reveal}}(i)$: On input i, if there was an i-th KeyGen query, then $\mathcal{E}$ corrupts P_i, gets the secret key $\mathsf{sk}_{\mathbb{A}_i}$, stores it, and outputs $\mathsf{sk}_{\mathbb{A}_i}$ to F. If there was no i-th KeyGen query $\mathcal{E}$ ignores the Reveal query.

$\mathcal{O}^{\text{Sign}}_{\mathsf{pp},\mathsf{msk}}(i, m_j, \mathbb{P}_j)$: On j-th query, if the corresponding secret key $\mathsf{sk}_{\mathbb{A}_i}$ was generated in a KeyGen query and not revealed, $\mathcal{E}$ activates P_i with $(\text{Signature}, \text{sid}, \mathsf{pp}, \mathbb{A}_i, m_j, \mathbb{P}_j)$. Eventually $\mathcal{E}$ gets $(\text{Signature}, \text{sid}, \mathbb{A}_j, m_j, \mathbb{P}_j, \sigma_j)$ back and returns σ_j to F. If secret key $\mathsf{sk}_{\mathbb{A}_i}$ was not generated in a KeyGen query ignore this signature query. In the case that $\mathsf{sk}_{\mathbb{A}_i}$ was revealed, the corresponding party was corrupted and $\mathcal{E}$ generates the signature with $\sigma_j \leftarrow \text{Sign}(\mathsf{pp}, \mathsf{sk}_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$ and returns it to F.

Eventually F outputs a triple $(m^*, \mathbb{P}^*, \sigma^*)$. If $(m^*, \mathbb{P}^*)$ was signed in a query or policy $\mathbb{P}^*$ accepts any $\mathbb{A}_i$ where the corresponding secret key $\mathsf{sk}_{\mathbb{A}_i}$ was revealed to F, $\mathcal{E}$ outputs 0 and aborts. Else $\mathcal{E}$ activates a new and not corrupted party P_V with $(\text{Verify}, \text{sid}, \mathsf{pp}, m^*, \mathbb{P}^*, \sigma^*)$ and outputs the result.

Analysis: For analyzing $\mathcal{E}$'s success we only have to consider its final output.

- If $\mathcal{E}$ interacts with $\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}^{(\cdot)}\right]$ every Verification activation for a signature under fixed public parameter pp, where the signature was not generated by the functionality, results in output 0, because $\mathcal{F}_{\text{ABS}}$ guarantees unforgeability. This is the case if the last step of $\mathcal{E}$ is reached. Hence, we get $\Pr\left[\mathcal{E}\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}^{(\cdot)}\right](\lambda) = 1\right] = 0$.
- If $\mathcal{E}$ interacts with $[\pi_{\text{ABS}}, \mathcal{A}]$, the view of forger F is distributed as in the experiment $\text{Exp}^{\text{UF}}_F(\lambda)$. We denote with "$F$ wins" the event $\text{Exp}^{\text{UF}}_F(\lambda) = 1$ and with "F fails" the event $\text{Exp}^{\text{UF}}_F(\lambda) = 0$. Accordingly, the following holds.

$$\begin{aligned}\Pr[\mathcal{E}[\pi_{\text{ABS}}, \mathcal{A}](\lambda) = 1] &= \Pr[\mathcal{E} \text{ outputs } 1 \mid F \text{ wins}] \cdot \Pr[F \text{ wins}] \\ &\quad + \Pr[\mathcal{E} \text{ outputs } 1 \mid F \text{ fails}] \cdot \Pr[F \text{ fails}] \\ &= 1 \cdot \tau(\lambda) + 0 \cdot (1 - \tau(\lambda)) \\ &= \tau(\lambda)\end{aligned}$$

Overall, we get that $|\Pr[\mathcal{E}[\pi_{\text{ABS}}, \mathcal{A}](\lambda) = 1] - \Pr\left[\mathcal{E}\left[\mathcal{F}_{\text{ABS}}, \mathcal{S}^{(\cdot)}\right](\lambda) = 1\right]| = \tau(\lambda)$, where $\tau(\lambda)$ is the non-negligible success probability of F in $\text{Exp}^{\text{UF}}_F(\lambda)$.

□

References

[AO12] Masayuki Abe and Miyako Ohkubo. "A framework for universally composable non-committing blind signatures". In: *IJACT* 2.3 (2012).

[AAS16] Hiroaki Anada, Seiko Arita, and Kouichi Sakurai. "Proof of Knowledge on Monotone Predicates and its Application to Attribute-Based Identifications and Signatures". In: *IACR ePrint* 2016 (2016), p. 483.

[Ate+05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. "Practical Group Signatures without Random Oracles". In: *IACR ePrint* 2005 (2005). URL: `http://ia.cr/2005/385`.

[BF14] Mihir Bellare and Georg Fuchsbauer. "Policy-Based Signatures". In: *Public Key Cryptography*. Vol. 8383. LNCS. Springer, 2014, pp. 520–537.

[BGK06] Aslak Bakke Buan, Kristian Gjøsteen, and Lillian Kråkmo. "Universally Composable Blind Signatures in the Plain Model". In: *IACR ePrint* 2006 (2006). URL: `http://ia.cr/2006/405`.

[Cam+15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. "Composable and Modular Anonymous Credentials: Definitions and Practical Constructions". In: *ASIACRYPT (2)*. Vol. 9453. LNCS. Springer, 2015, pp. 262–288.

[Cam+16] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. "Universal Composition with Responsive Environments". In: *ASIACRYPT (2)*. Vol. 10032. LNCS. 2016, pp. 807–840.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *FOCS*. IEEE Computer Society, 2001, pp. 136–145.

[Can03] Ran Canetti. "Universally Composable Signatures, Certification and Authentication". In: *IACR ePrint* 2003 (2003). URL: `http://ia.cr/2003/239`.

[Can13] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *IACR ePrint* 2013 (2013). URL: `http://ia.cr/2000/067`.

[EHM11] Alex Escala, Javier Herranz, and Paz Morillo. "Revocable Attribute-Based Signatures with Adaptive Security in the Standard Model". In: *AFRICACRYPT 2011*. Vol. 6737. LNCS. Springer, 2011, pp. 224–241.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM J. Comput.* 17.2 (1988), pp. 281–308.

[Her16] Javier Herranz. "Attribute-based versions of Schnorr and ElGamal". In: *Appl. Algebra Eng. Commun. Comput.* 27.1 (2016).

[Lin11] Yehuda Lindell. "Highly-Efficient Universally-Composable Commitments Based on the DDH Assumption". In: *EUROCRYPT*. Vol. 6632. LNCS. Springer, 2011, pp. 446–466.

[MPR08] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. "Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resistance". In: *IACR ePrint* 2008 (2008). URL: `http://ia.cr/2008/328`.

[MPR11] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. "Attribute-Based Signatures". In: *CT-RSA*. Vol. 6558. Lecture Notes in Computer Science. Springer, 2011, pp. 376–392.

[OT14] Tatsuaki Okamoto and Katsuyuki Takashima. "Efficient Attribute-Based Signatures for Non-Monotone Predicates in the Standard Model". In: *IEEE Trans. Cloud Computing* 2.4 (2014).

[SAH16] Yusuke Sakai, Nuttapong Attrapadung, and Goichiro Hanaoka. “Attribute-Based Signatures for Circuits from Bilinear Map”. In: *Public Key Cryptography (1)*. Vol. 9614. LNCS. Springer, 2016, pp. 283–300.