Policy-Compliant Signatures

Christian Badertscher[1], Christian Matt[2], and Hendrik Waldner[3]

[1] IOHK
christian.badertscher@iohk.io
[2] Concordium
cm@concordium.com
[3] University of Edinburgh
hendrik.waldner@ed.ac.uk

Abstract. We introduce *policy-compliant signatures (PCS)*. A PCS scheme can be used in a setting where a central authority determines a global policy and distributes public and secret keys associated with sets of attributes to the users in the system. If two users, Alice and Bob, have attribute sets that jointly satisfy the global policy, Alice can use her secret key and Bob's public key to sign a message. *Unforgeability* ensures that a valid signature can only be produced if Alice's secret key is known and if the policy is satisfied. *Privacy* guarantees that the public keys and produced signatures reveal nothing about the users' attributes beyond whether they satisfy the policy or not. PCS extends the functionality provided by existing primitives such as attribute-based signatures and policy-based signatures, which do not consider a designated receiver and thus cannot include the receiver's attributes in the policies. We describe practical applications of PCS which include controlling transactions in financial systems with strong privacy guarantees (avoiding additional trusted entities that check compliance), as well as being a tool for trust negotiations.

We introduce an indistinguishability-based privacy notion for PCS and present a generic and modular scheme based on standard building blocks such as signatures, non-interactive zero-knowledge proofs, and a (predicate-only) predicate encryption scheme. We show that it can be instantiated to obtain an efficient scheme that is provably secure under standard pairing-assumptions for a wide range of policies.

We further model PCS in UC by describing the goal of PCS as an enhanced ideal signature functionality which gives rise to a simulation-based privacy notion for PCS. We show that our generic scheme achieves this composable security notion under the additional assumption that the underlying predicate encryption scheme satisfies a stronger, fully adaptive, simulation-based attribute-hiding notion.

1 Introduction

Digital signatures provide authenticity to messages in the sense that everyone can verify that a signed message was indeed signed by a specific sender, and

not modified afterwards. Attribute-based signatures [26] and policy-based signatures [4] extend this concept by introducing policies that the sender needs to satisfy to generate a valid signature. We take this one step further and introduce *policy-compliant signatures (PCS)* with policies that take into account attributes of both, the sender and the receiver. This is useful in settings where messages have a designated receiver. A prevalent example of such a setting are blockchain applications, in which a sender signs a transaction sending funds to a given receiver. If such a system is used within a corporation and PCS are used for generating these signatures, the company can set a policy, restricting who can send funds to whom.

In more detail, a PCS scheme allows a central authority to generate a master public key and a master secret key for a given policy. The authority can then use the master secret key to generate public/private key pairs associated with a set of attributes. The signer Alice then uses her private signing key and the receiver Bob's public key to create a signature for a message. The signature can be publicly verified using all public keys. It is only valid if Alice's and Bob's attributes together satisfy the global policy.

Security requirements. Unforgeability of ordinary signature schemes ensures that valid signatures cannot be produced without knowledge of the secret key, and that signed messages cannot be modified without invalidating the signature. The unforgeability notion of PCS additionally requires that even with access to the secret key, it should not be possible for a malicious sender to craft a valid signature if the policy is not satisfied by the sender and the receiver.

In addition to unforgeability, PCS provide privacy for the sender's and receiver's attributes. Our privacy notion captures three different attack scenarios: First, outsiders only seeing the public keys and signatures between two parties should not learn anything about the attributes of these parties beyond the fact whether they satisfy the policy. Secondly, a (possibly malicious) sender should not learn anything about the receiver's attributes except whether their attributes satisfy the policy. And finally, a (possibly malicious) receiver should not learn anything about the sender's attributes except whether their attributes satisfy the policy.

The core challenge to obtain PCS. Consider the following attempt to obtain the functionality of a PCS scheme: A central authority is in charge of checking compliance of every single transaction by ensuring that whenever a sender S with attributes x sends a message to a receiver R with attributes x^*, the policy specified by $F(x, x^*)$ is satisfied. While conceptually simple, it does not satisfy our needs: One goal of PCS is to avoid a central authority assisting in the signature generation and verification because this results in a central point of failure in the execution of the system. Stated differently, the authority shall only be used to issue the credentials but the (non-interactive) signature generation and verification must be possible only with the public values associated to the receiver and the secret values of the sender.

In a second attempt, we let the authority issue ordinary signature key pairs (pk, sk) and a certificate of the respective attributes C_x to each participant in the system. To send a message m, a sender S signs the message m and proves, using a non-interactive zero-knowledge proof, that the attributes associated with the certificates of the sender and the receiver satisfy the policy $F(x, x^*)$. This second attempt looks more appealing, but it has the drawback that the sender must be aware of the recipient's attributes since otherwise no proof can be generated about the compliance with attributes not owned by the sender—especially if the certificate C_{x^*} is supposed to (computationally) hide the attributes of the receiver.[4]

We see that the main challenge to obtain PCS is to ensure that only valid signatures can be generated by a sender without a trusted authority assisting in the signature generation while the attributes of any entity in the system are hidden at any time, even from the sender. At first sight, this appears contradictory as it excludes any solution where the sender "proves" a joint statement including a receiver, using only public information about the receiver, which hides the receiver's attributes. The key idea to overcome this issue is to employ a specific form of predicate encryption that allows every participant to only learn a single bit of information upon generating a signature. This single leaked bit is $F(x, x^*)$ and the process does not leak anything beyond this evaluation. We additionally show that this specific form of predicate encryption is in fact *necessary* to obtain PCS.

1.1 Applications of PCS

Applications to financial payment systems. PCS can be used in all settings in which messages are sent to designated receivers and a global policy about the senders and receivers needs to be publicly verifiable. This naturally occurs in financial transactions, such as paying online services when purchasing, for example, digital content (such as movies) or services (such as online games or lotteries) that are region-dependent or age restricted. Typically, such services require additional authentication upon payment such as identity card information through scanning or manual input. PCS signatures can merge the act of authentication with the basic task of signing a transaction. A policy can be expressed as a list of requirements for say n categories of services S_i. For age and/or country restrictions, a policy might be given by $(\text{Age} \geq 18 \wedge S_1) \vee (\text{Age} \geq 16 \wedge \text{Country} = \text{CH} \wedge S_2) \vee \ldots$. Assume Alice obtained a key-pair from a credential management entity that is tied to her country of residence (akin to obtaining an ID card), and each service of Bob is assigned the correct category (identified also by a PCS public key for credential S_i). Then the payment system needs no additional check of the policy if the transactions are signed using a PCS scheme. If a transaction is successful,

[4] For the same reason, attempts to derive PCS in a black-box way from existing policy-based primitives fail (cf. Section 1.3) because they would require to implement a policy only based on the public key of the receiver, which does not allow to efficiently obtain their attributes.

(an honest) Bob can be sure that the client had access to appropriate private credentials. Thanks to the public verifiability, the transaction can be validated by an external auditor and by the attribute hiding property of PCS the exact combination of client attributes and service is not leaked (to the auditor) by the signature system.

Furthermore, in blockchain systems such as Bitcoin [28], a transaction transferring funds from a sender Alice to a receiver Bob contains a signature from Alice on the transaction details. Before adding such a transaction to a new block, the miners verify the validity of the transaction including the signature. When the used signature scheme is replaced by a PCS scheme, such transactions are only valid if the global policy allows Alice to send funds to Bob. This can be useful if the blockchain is used in a corporate environment where the money flow needs to be restricted in certain ways, e.g., defined by a legal system. Imagining a toy example, one could define a new company-wide digital token T with address format $addr = (pk_{\text{pcs}}, \ldots)$. A transaction transferring tokens T from $addr_A$ to $addr_B$ can only be valid if a (publicly verifiable) PCS signature confirms this transaction. By issuing credentials to employees and to facilities (such as canteens) within the company, and defining the policy to steer token flow (e.g., employees are allowed to exchange tokens or consume the tokens at company facilities), such tokens can be bound to a specific purpose at the sole cost of having to verify PCS signatures and address formats. The security of PCS makes it impossible for any sender to violate the company policy, both by accident or malice. This renders other compliance checks for this policy obsolete, such as techniques that are only triggered after suspicious transactions are observed and that often result in a complete revocation of a user's privacy [9, 15]. The attribute-hiding property of PCS further ensures that no information about the attributes of the transacting entities beyond that they satisfy the policy is revealed by the signatures and addresses (in the above toy example, we would not reveal whether it is a transaction between employees or between an employee and a facility). Thanks to this, the pseudonymity of the used blockchain system is preserved.

Applications to trust negotiations. Another application of PCS are trust-negotiation systems [19, 25]. Assume Alice and Bob work for an intelligence agency and need to exchange secret information. Further assume these agencies have a policy on who is allowed to exchange information with whom, e.g., based on the divisions and ranks of the involved parties as in role-based access control systems. In [25], the example assumes Alice has top-level clearance and before sending a message M, she must make sure that Bob also has top-level clearance. In the language of [25], what PCS brings to this setting is a simple implementation of the following two-party protocol: The common input are the access-control policy F (defined on the space of party credentials), and the agency's public parameters pp_{agency} (equivalent to a company-wide public-key infrastructure). Alice's private inputs are her message M and her credentials $cred_A$, and Bob's private input is his credentials $cred_B$. The output out_A of Alice and out_B of Bob are defined to

be

$$out_A = \begin{cases} 1, \text{ if } F(cred_A, cred_B) \\ 0, \text{ otherwise} \end{cases} \qquad out_B = \begin{cases} M, \text{ if } F(cred_A, cred_B) \\ \perp, \text{ otherwise} \end{cases} .$$

Assuming the agency has set up the public-key infrastructure, the above functionality is realized as follows: Alice encrypts the message M with Bob's (encryption) public key and signs the corresponding ciphertext with a PCS scheme (using her secret signing key, and Bob's signature public key). If the resulting signature is valid, then Alice sends the packet to Bob and otherwise does not send the message. If the policy is satisfied, then Bob learns the message. Otherwise, Bob learns nothing. The PCS scheme itself does not leak anything beyond the fulfillment of the policy.

1.2 Our Contributions and Organization of this Paper

PCS Notion. As a conceptual contribution, we introduce the notion of PCS (see Section 3). In addition to the syntactical requirements, we define unforgeability (in Section 3.2). This includes policy enforcement, i.e., unforgeability ensures that a signature that verifies with respect to the public verification key of the sender A and the receiver B can only be produced when possessing the secret signing key of A and if the attributes of A and B satisfy the policy.

Furthermore, we define an indistinguishability-based attribute hiding notion (in Section 3.3). This notion intuitively guarantees that an adversary cannot distinguish public keys and signatures generated for different sets of attributes, as long as the policy does not separate them.

Generic construction and concrete instantiation. We first provide an efficient generic construction of PCS from standard tools using digital signatures, (predicate-only) predicate encryption, and NIZK in Section 4. We show that relying on predicate-only PE is a tight fit for our goal in the sense that any PCS scheme gives rise to a related PE scheme. This settles an important feasibility question regarding constructions and efficiency for PCS in general.

Our generic construction is not only theoretically interesting, it also admits efficient instantiations (w.r.t. the indistinguishability-based attribute-hiding notion) based on standard pairing assumptions coupled with Groth-Sahai proofs for the rich class of predicates expressible by inner-products [22]. The policies that are realizable on top of the inner-product functionality range from CNF formulas and exact threshold clauses (with conjunctive or disjunctive clauses) to hidden-vector-encryption which in turn opens up the field for PCS to efficiently implement subset predicates, comparison predicates and their conjunctions as defined in [8].

Composable PCS and SIM-based notion. Finally, we cast PCS as an ideal, enhanced signature functionality in the spirit of [2,11] to model the ideal composable guarantees of PCS. We then derive a simpler simulation-based attribute hiding

notion (in Section 5.1) and prove that an unforgeable and sim-based secure PCS scheme realizes the ideal signature functionality. By definition of the ideal system, the sim-based notion guarantees that everything an attacker can learn from the public keys and signatures can be efficiently produced by a simulator given only the public information and the information for which signatures the policy is satisfied. This allows to capture precisely which information is leaked by a PCS scheme. We show that our generic construction achieves this notion if the underlying PE scheme satisfies a related (fully adaptive) simulation-based notion, which is stronger than what has been considered in the literature so far, notably in [17].

1.3 Related Work

We provide an overview of cryptographic primitives which have been introduced in the context of attribute-based and policy-dependent constructions to shed light on the role and necessity of PCS in this space.

Attribute-based signatures and policy-based signatures. Attribute-based signatures (ABS) [26] have similar goals to PCS: In an ABS scheme, an authority can generate secret signing keys associated to a set of attributes. The signer can then sign messages for some policy and the resulting signature is only valid if the signer's attributes satisfy the policy. Policy-based signatures [4] generalize this concept by allowing the policies to depend not only the sender's attributes but also on the signed messages. A clear distinction from PCS is that they do not allow the policies to depend on the receiver's attributes. Thus, the notions and security guarantees are very different.

Another difference between PCS and ABS is that an ABS scheme allows the sender to choose the policy for each message at the time of signing, whereas the policy in PCS schemes is fixed by the authority during the setup. This gives ABS more flexibility. Note, however, that allowing the sender to choose the policy in PCS schemes would be detrimental to our privacy guarantees: We want to protect the receiver's attributes even from malicious senders. Allowing the sender to choose many different policies and then verify the resulting signatures would allow a malicious sender to find the precise attributes of all receivers.

Finally, ABS provide an additional security guarantee that PCS do not offer, namely unlinkability of signatures. That is, given two signatures, one cannot determine whether they have been produced by the same signer; one only learns that somebody satisfying the policies signed. In a PCS scheme, this is not required since it is not needed for the applications we have in mind. For example, when used in a blockchain system providing pseudonymity, the signatures are anyway linked to the pseudonyms of the senders and receivers of transactions. Trying to hide the signer would thus not be useful in this context.

Designated verifier signatures. Designated verifier signatures have been introduced by Jakobsson et al. [21]. As in our setting, they consider signatures produced for a designated receiver. They require that only this receiver can verify the

signatures. Furthermore, the receiver should not be able to convince others of the validity of such signatures. This is in contrast to PCS, which can be verified publicly. The setting and security requirements are thus very different.

Matchmaking Encryption. The high-level goals of PCS and matchmaking encryption (ME) introduced by Ateniese et al. [1] seem similar, but turn out to be quite distinctive due to the respective applications in mind. ME captures a non-interactive variant of a secret-handshake (with payload), that is, in addition to the functionality that PCS supports. In ME, the sender has the freedom to define the receiver's policy and the receiver can in addition to its private key (for the attributes), receive an additional policy decryption key that captures a policy on the sender's attributes under which the receiver is able to decrypt the ciphertext. These two receiver private keys can conceptually be merged into one single attribute-policy decryption key, which results in a seemingly simpler notion that is realizable from standard FE (capturing the policy as a specific function). This notion is dubbed arranged ME (A-ME).

In a nutshell, our unforgeability requirements are stronger and require that even if sender and receiver collude, they should not be able to produce a valid (publicly verifiable) signature (authenticity of ME is a guarantee for an honest receiver not to be fooled by a ciphertext of a sender that does not possess the required attributes). Second, the ME authenticity game does not provide an oracle to the adversary for computations on the private key, therefore disallowing all attacks that are based on malleable ciphertexts, which is problematic for our needs. This aspect also influences the obtained privacy guarantees. In the ME security game, the adversary only obtains a single value (the ciphertext) that is a function of the sender's secret key. For ME, this makes a lot of sense as it is used to replace a handshake with a single payload message. We, however, need a signing oracle and hence obtain strictly stronger privacy. For the sake of self-containment, we sketch an (A-)ME scheme which does not provide the attribute hiding property of PCS in the full version.

Finally, constructions of PCS for simple policies like CNF, conjunctions of equalities or comparisons, are in the standard model and have practical instantiations. In contrast, even for simple equality policies where the FE and randomized FE are not needed as building blocks, the constructions of [1] are in the random oracle model.

Access control encryption. The notion of access control encryption (ACE) [3,16] is a cryptographic primitive that allows to control the information flow within a system. ACE is not suitable to achieve the task we need. First, the system relies crucially on a third-party called the sanitizer which is a role that does not fit into our setting. Secondly, ACE only protects the information flow within the system (when running through a sanitizer), whereas in our system, corrupted parties might meet offline trying to generate a valid joint signature, which must be part of the attack model.

Predicate encryption and attribute-based encryption. Predicate encryption and attribute-based encryption allow decryption of ciphertexts only for users with secret keys matching a certain policy. While PCS are signatures and not encryption schemes, they are still related because of the required privacy notion. In particular, our indistinguishability-based and simulation-based attribute hiding properties are closely related to the respective notions for these encryption schemes.

The notion of predicate encryption has first been considered in [8, 22]. In the work of Boneh and Waters [8], the authors construct a scheme that allows for comparison, subset and arbitrary conjunctive queries. In the succeeding work of Katz et al. [22], the authors present a scheme for the inner product functionality and the authors also observe that the inner product functionality is sufficient for polynomial predicate evaluations as well as DNF and CNF formulas. We mention more regarding the common policies of these schemes below. Since the results of Boneh and Waters [8] and Katz et al. [22], more works for the same functionality class have been proposed [29, 30], as well as for the stronger notion of partially-hiding predicate encryption [17, 31]. Partially-hiding predicate encryption is a generalization of predicate encryption in which the ciphertext is extended with public attributes. The function associated with the functional key is then first applied on the public information and the result is then used together with hidden attribute of the ciphertext.

2 Preliminaries

We denote the security parameter with $\lambda \in \mathbb{N}$ and use 1^λ as its unary representation. We call a randomized algorithm $\mathcal{A}$ *probabilistic polynomial time* (PPT) if there exists a polynomial $p(\cdot)$ such that for every input x the running time of $\mathcal{A}(x)$ is bounded by $p(|x|)$. A function $\mathrm{negl} : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* if for every positive polynomial $p(\lambda)$, there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0 \colon \mathrm{negl}(\lambda) < 1/p(\lambda)$. If clear from the context, we sometimes omit λ for improved readability. The set $\{1, \ldots, n\}$ is denoted as $[n]$ for $n \in \mathbb{N}$. For the equality check of two elements, we use "=". The assign operator is denoted with ":=", whereas randomized assignment is denoted with $a \leftarrow A$, with a randomized algorithm A and where the randomness is not explicit. If the randomness is explicit, we write $a := A(x; r)$ where x is the input and r is the randomness. For algorithms $\mathcal{A}$ and $\mathcal{B}$, we write $\mathcal{A}^{\mathcal{B}(\cdot)}(x)$ to denote that $\mathcal{A}$ gets x as an input and has oracle access to $\mathcal{B}$, that is, the response for an oracle query q is $\mathcal{B}(q)$.

Further preliminaries on digital signature schemes, non-interactive zero-knowledge proofs and predicate encryption can be found in the full version.

3 Policy-Compliant Signatures

In this section, we introduce the notion of policy-compliant signature (PCS) schemes together with the notion of unforgeability and indistinguishability-based attribute hiding. We start by describing the syntax of PCS schemes, which

consists of four algorithms, responsible for the setup of the parameters, the key generation and the signature generation and verification.

Definition 3.1 (Policy-Compliant Signatures). *Let $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of attribute sets and denote by $\mathcal{X}_\lambda$ the powerset of X_λ. Further let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of sets $\mathcal{F}_\lambda$ of predicates $F\colon \mathcal{X}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}$. Then a* policy-compliant signature (PCS) *scheme for the functionality class $\mathcal{F}_\lambda$ is a tuple of four PPT algorithms* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*:*

$\mathsf{Setup}(1^\lambda, F)$**:** *On input a unary representation of the security parameter λ and a policy $F \in \mathcal{F}_\lambda$, output a master public and secret key pair* $(\mathsf{mpk}, \mathsf{msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, x)$**:** *On input the master secret key* msk *and a set of attributes $x \in \mathcal{X}_\lambda$, output a public and secret key pair* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}_S, \mathsf{pk}_R, m)$**:** *On input the master public key* mpk*, a sender secret key* sk_S*, a receiver public key* pk_R *and a message m, output either a signature σ or $\perp$.*

$\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$**:** *On input the master public key* mpk*, a sender public key* pk_S*, a receiver public key* pk_R*, a message m and a signature σ, output either 0 or 1.*

A Policy-Compliant Signature scheme is called correct*, if for all messages m, policies $F \in \mathcal{F}_\lambda$, and sets of attributes $x_1, x_2 \in \mathcal{X}_\lambda$, for all pairs* $(\mathsf{mpk}, \mathsf{msk})$ *in the support of* $\mathsf{Setup}(1^\lambda, F)$*, all key pairs* $(\mathsf{pk}_S, \mathsf{sk}_S)$ *and* $(\mathsf{pk}_R, \mathsf{sk}_R)$ *in the corresponding support of* $\mathsf{KeyGen}(\mathsf{msk}, x_1)$ *and* $\mathsf{KeyGen}(\mathsf{msk}, x_2)$*, respectively,*

$$\Pr\left[\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S, \mathsf{pk}_R, m, \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}_S, \mathsf{pk}_R, m)) = F(x_1, x_2)\right] \geq 1 - \mathrm{negl}(\lambda),$$

where the probability is over the random coins of Sign *and* Verify*.*

3.1 Adversarial Capabilities in the Security Games

Before diving into the security properties, we briefly explain the adversarial capabilities. The adversary can (using the oracle $\mathsf{QKeyGen}$ or $\mathsf{QKeyGenLR}$) obtain public keys for chosen attributes, which models honest parties in the system of which the public key is known; (using the oracle QCor) obtain the secret key corresponding to a given public key, which models the adversary corrupting a party; and (using the oracle QSign) obtain signatures relative to chosen public keys, which models the adversary seeing signatures from honest parties.

More formally, in a context where a master secret key msk is defined (as will be the case in our security experiments), we capture the above by defining the following stateful oracles that maintain the initially empty sets $\mathcal{QK}$, $\mathcal{QC}$, and $\mathcal{QS}$.

Key-Generation Oracle $\mathsf{QKeyGen}(\cdot)$**:** On the ith input of an attribute set x_i, generate $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x_i)$, add $(i, \mathsf{pk}_i, \mathsf{sk}_i, x_i)$ to $\mathcal{QK}$, and return pk_i.

Left-or-Right Key-Generation Oracle $\mathsf{QKeyGenLR}(\cdot,\cdot)$**:** On the ith input of a pair of attribute sets $x_{i,0}$ and $x_{i,1}$, generate $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x_{i,\beta})$, add $(i, \mathsf{pk}_i, \mathsf{sk}_i, x_{i,0}, x_{i,1})$ to $\mathcal{QK}$, and return pk_i. In this case, the bit β is defined by the security game.

Corruption Oracle $\mathsf{QCor}(\cdot)$**:** On input an index i, if $\mathcal{QK}$ contains an entry $(i, \cdot, \mathsf{sk}_i, \cdot) \in \mathcal{QK}$ or $(i, \cdot, \mathsf{sk}_i, \cdot, \cdot) \in \mathcal{QK}$ for some sk_i, then add that entry from $\mathcal{QK}$ to $\mathcal{QC}$ and return sk_i. Otherwise, return $\perp$.

Signing Oracle $\mathsf{QSign}(\cdot,\cdot,\cdot)$**:** On input a (sender) index i, a (receiver) public key pk', and a message m, if $\mathcal{QK}$ contains an entry $(i, \mathsf{pk}_i, \mathsf{sk}_i, \cdot) \in \mathcal{QK}$ or $(i, \mathsf{pk}_i, \mathsf{sk}_i, \cdot, \cdot) \in \mathcal{QK}$ for some pk_i and sk_i, then return $\sigma \leftarrow \mathsf{PCS.Sign}(\mathsf{mpk}, \mathsf{sk}_i, \mathsf{pk}', m)$ and add $(i, \mathsf{pk}_i, \mathsf{pk}', m, \sigma)$ to $\mathcal{QS}$. Otherwise, return $\perp$.

3.2 Existential Unforgeability

The unforgeability notion captures that an adversary $\mathcal{A}$ is not able to create a valid signature for a public key that belongs to an uncorrupted party. Additionally, the adversary should also not be able to create a valid signature for a pair of public keys that do not fulfill the policy. More precisely, any signature for a new message m^* that successfully verifies, with respect to arbitrary sender and receiver public keys, constitutes a forgery unless the adversary has obtained the private key corresponding to the public key associated to the sender's attribute set x, and the receiver public key is associated to attribute set x^* obtained via the key generation oracle, and $F(x, x^*) = 1$. An interesting special case is regarding collisions of public keys. Here a forgery is valid unless the adversary has corrupted all indexes i corresponding to that public key.[5] Note that as a further special case that the adversary cannot create a valid signature w.r.t. public keys that have not been output by the key generation authority (formally, the condition on the last line in Fig. 1 is trivially true). Looking ahead, this game-based notion in fact captures all unforgeability properties we motivated for PCS: we show in Section 5 that Definition 3.2 implies ideal unforgeability properties when modeling PCS as an enhanced signature functionality.

We capture these requirements using an existential unforgeability game:

Definition 3.2 (Existential Unforgeability of a PCS Scheme). *Let* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a PCS scheme as defined in Definition 3.1. We define the experiment* $\text{EUF-CMA}^{\mathsf{PCS}}$ *in Fig. 1 and define the advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *by*

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{PCS},\mathcal{A}}(\lambda) = \Pr[\text{EUF-CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}) = 1].$$

A PCS scheme PCS *is called* existential unforgeable under adaptive chosen message attacks *or* existential unforgeable *for short if for any polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{PCS},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda)$.

[5] This is vital to our use case of PCS: as long as a given user is not corrupted, no one is able to produce valid signatures that could be considered valid signatures of that party.

$\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$

$(F, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, F)$

$(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) \leftarrow \mathcal{A}_2^{\mathsf{QKeyGen}(\cdot),\mathsf{QCor}(\cdot),\mathsf{QSign}(\cdot,\cdot,\cdot)}(\mathsf{st}, \mathsf{mpk})$

Output: $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge$

$$\Big[\big[\exists(\cdot, \mathsf{pk}, \cdot, \cdot) \in \mathcal{QK} \setminus \mathcal{QC} \wedge (\cdot, \mathsf{pk}, \mathsf{pk}^*, m^*, \cdot) \notin \mathcal{QS}\big] \vee \forall(i, \mathsf{pk}, \cdot, x_i), (\cdot, \mathsf{pk}^*, \cdot, x^*) \in \mathcal{QK} : F(x_i, x^*) = 0\Big]$$

Fig. 1: Unforgeability Game of PCS.

3.3 Indistinguishability-Based Attribute Hiding

We formalize the notion of attribute hiding as a security game. In this security game, the adversary has access to a left-or-right key-generation oracle that it can query multiple times using pairs of attribute sets (x_0, x_1) to obtain the key for x_β, where β is a random bit sampled in the beginning of the game. The goal of the adversary is to guess the bit β. To achieve this, it additionally has access to a corruption oracle with with it can obtain the secret keys corresponding to previously obtained public keys. This is only allowed for public keys that previously have been generated for the same attribute set, i.e. $x_0 = x_1$. Furthermore, the adversary is also allowed to query a signing oracle to obtain signatures generated for sender and receiver key pairs of its choice.

To prevent the adversary from trivially distinguishing between the generated public keys, we need to exclude two kinds of trivial attacks: first, if x_β is seen as the receiver attributes, then distinguishing is trivial if the adversary possesses a secret key for the attribute set x such that $F(x, x_\beta) \neq F(x, x_{1-\beta})$. Second, if a signing query is asked for a pair of challenge keys such that $F(x_\beta, x'_\beta) \neq F(x_{1-\beta}, x'_{1-\beta})$, where x_β and $x_{1-\beta}$ are the attribute sets potentially associated with the sender key and x'_β and $x'_{1-\beta}$ are the attribute sets potentially associated with the receiver key, then distinguishing is trivial. Any other interaction is deemed valid.

Definition 3.3 (IND-Based Attribute Hiding). *Let* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a PCS scheme as defined in Definition 3.1. For* $\beta \in \{0, 1\}$*, we define the experiment* $\mathrm{AH}_\beta^{\mathsf{PCS}}$ *in Fig. 2, where all oracles are defined as in Section 3.1. The advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is defined by*

$$\mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PCS},\mathcal{A}}(\lambda) = |\Pr[\mathrm{AH}_0^{\mathsf{PCS}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathrm{AH}_1^{\mathsf{PCS}}(1^\lambda, \mathcal{A}) = 1]|.$$

We call an adversary valid *if all of the following hold with probability* 1 *over the randomness of the adversary and all involved algorithms:*

- *for every* $(i, \cdot, \cdot, x_{i,0}, x_{i,1}) \in \mathcal{QC}$ *and for all* $(\cdot, \cdot, \cdot, x_{j,0}, x_{j,1}) \in \mathcal{QK}$*, we have* $x_{i,0} = x_{i,1} =: x_i$ *and* $F(x_i, x_{j,0}) = F(x_i, x_{j,1})$*,*

– *and for all* $(i, \cdot, \mathsf{pk}_j, \cdot, \cdot) \in \mathcal{QS}$*, and* $(i, \cdot, \cdot, x_{i,0}, x_{i,1}), (\cdot, \mathsf{pk}_j, \cdot, x_{j,0}, x_{j,1}) \in \mathcal{QK}$*, we have* $F(x_{i,0}, x_{j,0}) = F(x_{i,1}, x_{j,1})$*.*

A PCS scheme PCS *is called* attribute hiding *if for any valid polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PCS},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$*.*

$\mathbf{AH}^{\mathsf{PCS}}_{\beta}(1^\lambda, \mathcal{A})$

$(F, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PCS.Setup}(1^\lambda, F)$

$\alpha \leftarrow \mathcal{A}_2^{\mathsf{QKeyGenLR}(\cdot,\cdot),\mathsf{QCor}(\cdot),\mathsf{QSign}(\cdot,\cdot,\cdot)}(\mathsf{st}, \mathsf{mpk})$

Output: α

Fig. 2: The Strong Attribute Hiding Game for PCS.

4 Construction of a Policy-Compliant Signature Scheme

We present in Section 4.1 our policy-compliant signature scheme, show that it is correct in Section 4.2, proof its security in Sections 4.3 and 4.4, and show in Section 4.5 how the scheme, which is quite generic, can be instantiated from standard assumptions.

4.1 The Scheme

The high-level idea of the scheme is to let PCS signatures generated by the signer contain proofs that part of the target's public key can be decrypted. Recall that the challenge of our notion is to publish a *single* public-key that hides all attributes, but where *all* a priori legitimate parties can figure out the bit of information whether they jointly satisfy the policy. For this step, we use a predicate-only predicate encryption scheme for the specific functionality class induced by the policy. To allow for the evaluation of the global policy on the inputs of the sender and the receiver using a predicate encryption scheme, we define a deterministic encoding function $\mathsf{SubPol}(F, x) = (\mathsf{SubPol}_1(F, x), \mathsf{SubPol}_2(F, x))$ that takes as input the global policy F and a set of attributes x and outputs a subpolicy encoding $f_{\boldsymbol{x}}$ (output of SubPol_1) and the attribute encoding $\boldsymbol{x}$ (output of SubPol_2) for the associated PE scheme. Functionally, we have

$$\begin{aligned} &\mathsf{SubPol}(F, x) = (\mathsf{SubPol}_1(F, x), \mathsf{SubPol}_2(F, x)), \\ &\text{s.t. } \forall x, x' \in \mathcal{X} : F(x, x') = \underbrace{\mathsf{SubPol}_1(F, x)(\mathsf{SubPol}_2(F, x'))}_{=f_{\boldsymbol{x}}(\boldsymbol{x}')}. \end{aligned} \tag{1}$$

We note that the usage of PE is not a coincidence here as there is an interesting theoretical connection between PCS and PE which we give in the full version. To turn the scheme into a secure PCS scheme, we still need to protect the integrity which entails the binding of public-keys and the proof-of-decryption, as well as binding public keys to the authority. Here, we make use of two types of signatures, namely existentially unforgeable signatures as well as strongly unforgeable signatures. Finally, a NIZK proof is used to establish the core relation of Fig. 4 to prove the above binding and correct decryption.

The full scheme is given in Fig. 3. Later in Sections 4.3 and 4.4 we prove the concrete security of the scheme. The implied succinct asymptotic security statement can be stated as follows:

Theorem 4.1 (Security of our PCS Construction (Asymptotic version)). *The PCS scheme PCS in Fig. 3 (w.r.t. policies $F \in \mathcal{F}$) is unforgeable and attribute hiding, if the signature schemes $\mathsf{DS}_{\mathsf{priv}}$ and DS_{P} are unforgeable, the signature scheme $\mathsf{DS}_{\mathsf{pub}}$ is strongly unforgeable, PE is an attribute-hiding (predicate-only) predicate encryption scheme (for the induced predicates from Eq. (1)), and NIZK is a secure non-interactive zero-knowledge proof of knowledge system for the relation R_{ZK} of Fig. 4.*

4.2 Correctness

The correctness of the construction described in Fig. 3 follows from the correctness of the predicate encryption scheme, the signature schemes, and the non-interactive zero-knowledge proof. Note that for the sake of exposition, we assume perfect correctness. However, even if any of the underlying building blocks has negligible correctness failure, this propagates through our scheme and would make it violate correctness only with negligible probability. Consider any two attribute sets x, y with $F(x, y) = 1$ (the other case for $F(x, y) = 0$ is straightforward) and let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x)$, $(\mathsf{pk}_y, \mathsf{sk}_y) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, y)$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{pk}_y := (\mathsf{vk}_y, \mathsf{ct}_y, \sigma^y_{\mathsf{pub}}), \mathsf{sk}_x := (\mathsf{vk}_x, \mathsf{sk}^{\mathsf{DS}}_x, \mathsf{sk}_{f_x}, \sigma^x_{\mathsf{priv}}), m)$ for an arbitrary message m. We have $\sigma \neq \perp$ because the check during the signature generation whether $\mathsf{PE.Dec}(\mathsf{sk}_{f_x}, \mathsf{ct}_y) = 1$ will be satisfied for $F(x, y) = 1$ due to the correctness of the scheme PE and the requirement in Eq. (1). Furthermore, the signature on the sender's public key verifies by the correctness of the signature scheme $\mathsf{DS}_{\mathsf{pub}}$ during the signing process. In the signature verification step, the calls to Verify for the signatures schemes $\mathsf{DS}_{\mathsf{pub}}, \mathsf{DS}_{\mathsf{priv}}$ and DS_{P} always return 1 by the correctness of the signature schemes $\mathsf{DS}_{\mathsf{pub}}, \mathsf{DS}_{\mathsf{priv}}$ and DS_{P}. Furthermore, $\mathsf{NIZK.Verify}$ always returns 1 by the correctness of NIZK. This proves the correctness of the PCS scheme.

4.3 Existential Unforgeability

After showing the correctness of our construction, we prove its unforgeability.

$\underline{\mathsf{Setup}(1^\lambda, F)}$:
$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
$\mathsf{msk}_{\mathsf{PE}} \leftarrow \mathsf{PE.Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$
$\mathsf{mpk} := (F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}})$
$\mathsf{msk} := (\mathsf{msk}_{\mathsf{PE}}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}})$
Return $(\mathsf{mpk}, \mathsf{msk})$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, x)}$:
Parse $\mathsf{msk} := (\mathsf{msk}_{\mathsf{PE}}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}})$
$(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$
$(f_{\boldsymbol{x}}, \boldsymbol{x}) = \mathsf{SubPol}(F, x)$
$\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mathsf{msk}_{\mathsf{PE}}, \boldsymbol{x})$
$\mathsf{sk}_{f_{\boldsymbol{x}}} \leftarrow \mathsf{PE.KeyGen}(\mathsf{msk}_{\mathsf{PE}}, f_{\boldsymbol{x}})$
$\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}))$
$\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}))$
$\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, \sigma_{\mathsf{pub}})$
$\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$
Return $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, \mathsf{pk}_R, m)}$:
Parse $\mathsf{mpk} = (F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}})$
$\mathsf{sk} = (\mathsf{vk}_S, \mathsf{sk}_S, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$
$\mathsf{pk}_R = (\mathsf{vk}_R, \mathsf{ct}_R, \sigma_{\mathsf{pub}})$
If $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_R, \mathsf{ct}_R), \sigma_{\mathsf{pub}}) = 0$
Return $\perp$
If $\mathsf{PE.Dec}(\mathsf{sk}_{f_{\boldsymbol{x}}}, \mathsf{ct}_R) = 0$
Return $\perp$
$\pi \leftarrow \mathsf{Prove}(\mathsf{CRS}, (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_S, \mathsf{vk}_R, \mathsf{ct}_R), (\mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}}))$
with L defined corresponding to Fig. 4.
$\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}_S, (m, \pi))$
Return $\sigma := (\pi, \sigma')$

$\underline{\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)}$:
Parse $\mathsf{mpk} = (F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}})$
$\mathsf{pk}_S = (\mathsf{vk}_S, \mathsf{ct}_S, \sigma^S_{\mathsf{pub}})$
$\mathsf{pk}_R = (\mathsf{vk}_R, \mathsf{ct}_R, \sigma^R_{\mathsf{pub}})$
$\sigma = (\pi, \sigma')$
(Return 0 if parsing fails or $\sigma = \perp$)
Return
$\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_R, \mathsf{ct}_R), \sigma^R_{\mathsf{pub}})$
$\wedge \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_S, \mathsf{ct}_S), \sigma^S_{\mathsf{pub}})$
$\wedge \mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_R, \mathsf{vk}_S, \mathsf{ct}_R), \pi)$
$\wedge \mathsf{DS}_{\mathsf{P}}.\mathsf{Verify}(\mathsf{vk}_S, (\pi, m), \sigma')$

Fig. 3: The Policy-Compliant Signature Scheme. It uses a NIZK proof system NIZK, a predicate encryption scheme PE, and three digital signature schemes $\mathsf{DS}_{\mathsf{pub}}, \mathsf{DS}_{\mathsf{priv}}$ and DS_{P}.

Relation R_{ZK}:
Instance: $x = (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_S, \mathsf{vk}_R, \mathsf{ct}_R)$
Witness: $w = (\mathsf{sk}_{f_\infty}, \sigma_{\mathsf{priv}})$
$R_{\mathrm{ZK}}(x, w) = 1$ if and only if:
$\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}_S, \mathsf{sk}_{f_\infty}), \sigma_{\mathsf{priv}}) = 1$ and $\mathsf{PE.Dec}(\mathsf{sk}_{f_\infty}, \mathsf{ct}_R) = 1$

Fig. 4: Relation used for the PCS scheme in Fig. 3.

Theorem 4.2. *Let* $\mathsf{DS}_{\mathsf{pub}} = (\mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify})$ *be a* SUF-CMA *secure signature scheme and let* $\mathsf{DS}_{\mathsf{priv}} = (\mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify})$ *and* $\mathsf{DS}_{\mathsf{P}} = (\mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{P}}.\mathsf{Verify})$ *be a EUF-CMA secure signature scheme and let* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *be an extractable proof system, then the construction* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figure 3, is existentially unforgeable. Namely, for any PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}, \mathcal{B}', \mathcal{B}''$ *and* $\mathcal{B}'''$*, such that*

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{PCS},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}}(\lambda) + 2q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{P}},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}''}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}'''}(\lambda),$$

where q *denotes the number of queries to* $\mathsf{QKeyGen}$*.*

Proof (Sketch). To prove the unforgeability of our PCS scheme, we introduce several bad events and bound their respective probabilitises by the unforgeability of the different signature schemes as well as the extractability of the NIZK proof system.

The first event that we need to bound is the event that an adversary generates a valid key toegether with a valid signature, without querying the key generation oracle. This event cannot occur due to the strong unforgeability of the signature scheme $\mathsf{DS}_{\mathsf{pub}}$. We need strong unforgeability here to prevent an adversary from turning an existing key into a new key by generating a different, valid, signature for this key. The second event that we need to bound is the event that the adversary is able to generate a valid signature using an existing key for which it does not know the corresponding secret key. This event can directly be bounded by the existential unforgeability of the signature scheme DS_{P}. The third, and last, event that we need to bound is the event in which the adversary creates a valid signature for two keys that do not fulfill the policy. The occurrence of this event can be bound by the extractability of the NIZK proof system and the unforgeability of the signature scheme $\mathsf{DS}_{\mathsf{priv}}$ and DS_{P}. In more detail, if an adversay is able to create a valid signature for a key pair, where the corresponding attributes do not fulfill the policy, then it has either (1) generated a NIZK proof for an incorrect statement, which is a contradiction to the extractability of the NIZK proof system; (2) has generated a valid witness for the NIZK proof by forging a signature of the $\mathsf{DS}_{\mathsf{priv}}$ signature scheme. This event is a contradiction to the existential unforgeability of the $\mathsf{DS}_{\mathsf{priv}}$ signature scheme. The third, and

Game	CRS, π	pk	justification
G_0	$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$	$\mathsf{KeyGen}(\mathsf{msk}, x_0)$	
G_1	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{KeyGen}(\mathsf{msk}, x_0)$	Zero-knowledge of NIZK
G_2	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{KeyGen}(\mathsf{msk}, x_1)$	AH of PE
G_3	$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$	$\mathsf{KeyGen}(\mathsf{msk}, x_1)$	Zero-knowledge of NIZK

Fig. 5: Overview of the games to prove the indistinguishability of attribute-hiding of the policy-compliant signature scheme described in Fig. 3.

last, possiblity of the adversary to produce a forgery would be to if it obtained two public keys which allowed for a "mix and match" attack, which however is exlcuded by bounding the collision probability of the keys. The proof then follows by showing that if none of these bad events occur, then no PCS forgery exists.

The formal proof of this theorem can be found in the full version. □

4.4 Indistinguishability-Based Attribute Hiding

We next prove that our PCS scheme is attribute hiding.

Theorem 4.3. *Let* $\mathsf{PE} = (\mathsf{PE.Setup}, \mathsf{PE.KeyGen}, \mathsf{PE.Enc}, \mathsf{PE.Dec})$ *be a predicate encryption scheme, let* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *be a NIZK proof system and let* $\mathsf{DS}_{\mathsf{pub}} = (\mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify})$ *be a strongly unforgeable signature scheme, then the construction* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figure 3, is attribute hiding. Namely, for any valid PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}, \mathcal{B}'$ *and* $\mathcal{B}''$*, such that:*

$$\mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PCS},\mathcal{A}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PE},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}''}(\lambda).$$

Proof. To prove this statement, we use a hybrid argument with the games defined in Fig. 5. Note that G_0 corresponds to the game $\mathrm{AH}_0^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ and G_3 to the game $\mathrm{AH}_1^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$. This results in:

$$\begin{aligned}\mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PCS},\mathcal{A}}(1^\lambda) &= |\Pr[\mathrm{AH}_0^{\mathsf{PCS}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathrm{AH}_1^{\mathsf{PCS}}(1^\lambda, \mathcal{A}) = 1]| \\ &= |\Pr[\mathsf{G}_0(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{G}_3(\lambda, \mathcal{A}) = 1]|.\end{aligned}$$

We describe the different games in more detail:

Game G_1: In this game, we change from an honestly generated CRS and honestly generated proofs to a simulated CRS and simulated proofs. The transition from G_0 to G_1 is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\Pr[\mathsf{G}_0(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{G}_1(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK}, \mathcal{B}_0}(\lambda).$$

Game G_2: In this game, we change the attributes used for the generation of the challenge public keys pk_i from $x_{i,0}$ to $x_{i,1}$ for all i. The transition from G_1 to G_2 is justified by the attribute-hiding property of PE and the strong unforgeability of $\mathsf{DS}_{\mathsf{pub}}$. Intuitively, we rely on the strong unforgeability of the signature scheme $\mathsf{DS}_{\mathsf{pub}}$ to prevent an adversary from learning any information about the challenge keys by obtaining a signature for a maliciously generated key. Namely, we can exhibit PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

$$|\Pr[\mathsf{G}_1(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{G}_2(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PE}, \mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}}, \mathcal{B}_2}(\lambda).$$

Game G_3: This game is the $\mathrm{AH}_1^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ game. In this game, we change back from a simulated CRS and simulated proofs π to an honestly generated CRS and honestly generated proofs π. As the transition from G_0 to G_1, this transition is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $\mathcal{B}_3$ such that:

$$|\Pr[\mathsf{G}_2(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{G}_3(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK}, \mathcal{B}_3}(\lambda).$$

Putting everything together, we obtain the theorem. □

We present the proofs for the different transitions in the full version.

4.5 Efficient Instantiations based on Inner-Product PE

In this section, we show that our generic PCS scheme can be instantiated efficiently for certain policies such as the ones mentioned in the introduction. Since the most efficient predicate-only PE schemes are known for the inner-product functionality class in the standard model, we focus on this instantiation and briefly recall the associated realizable policies established in [8, 22]. The two functionality classes we recall are:

Inner-Product Functionality. The functionality class is defined as $\mathcal{F}^{\mathsf{IP}}_{N,k} = \{F^{\mathrm{IP}}_{N,k} : \mathbb{Z}_N^k \times \mathbb{Z}_N^k \to \{0,1\}\}$ by the equation

$$F^{\mathrm{IP}}_{N,k}(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0 \bmod N, \\ 0 & \text{if } \langle \boldsymbol{x}, \boldsymbol{y} \rangle \neq 0 \bmod N. \end{cases}$$

Hidden-Vector Functionality. Define $\Sigma_* = \Sigma \cup \{*\}$ with $\Sigma = \{0,1\}$. The functionality class is defined as $\mathcal{F}_k^{\mathsf{HV}} = \{F_k^{\mathrm{HV}} : \Sigma_*^k \times \Sigma^k \to \{0,1\}\}$ by the equation

$$F_k^{\mathrm{HV}}(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \forall i \in [k]\ (x_i = y_i \text{ or } x_i = *), \\ 0 & \text{otherwise.} \end{cases}$$

In the following, we call a predicate encryption scheme that implements the IP functionality inner-product encryption (IPE) and refer to a predicate encryption scheme that implements the HV functionality as hidden-vector encryption (HVE). Note that the predicates in the associated PE schemes correspond to the functions $F_{N,k}^{\mathrm{IP}}(\boldsymbol{x}, \cdot)$ and $F_k^{\mathrm{HV}}(\boldsymbol{x}, \cdot)$ parameterized by the vector $\boldsymbol{x}$ corresponding to the first argument of the above functions, respectively. As shown in [22], HVE with dimension ℓ can be realized generically based on IPE of dimension 2ℓ.

Instantiating the generic scheme. The elements of our generic construction are digital signatures, predicate encryption, and NIZK. For inner-product predicates there exist efficient PE schemes for the assumed indistinguishability-based security [29,30] (and also for a certain type of simulation-based security [17]). For the signature scheme used by the authority to generate σ_{pub} and the signature scheme used by the client, we can use BLS signatures [6] (or BB signatures [5] to avoid switching to an idealized model). For the signature scheme used by the authority to generate σ_{priv}, we however have to pay attention, as it is used as part of the witness in a NIZK computation. The only source of practical inefficiency comes from the additional usage of the NIZK proof for the relation $R_{\mathrm{ZK}}(x, w) \leftrightarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}_S, \mathsf{sk}_{f_{\mathsf{id}}}), \sigma_{\mathsf{priv}}) \wedge \mathsf{Dec}(\mathsf{sk}_{f_{\mathsf{id}}}, \mathsf{ct}_R) = 1$, as it combines a generic signature verification with a proof of decryption of the PE scheme. Note that there are two signature schemes involved: the signature scheme with which the authority produces σ_{priv} is the crucial one in this section. For the "inner signature" (the one used by a party to sign the final message) it will only be convenient to assume that vk_S is encoded as a group element of some cyclic group (which is the case for the variants discussed above). Note that the NIZK relation does not involve signatures of the inner scheme, just the representation of the public key as part of the statement.

To avoid the potential source of inefficiency from the NIZK we can use predicate encryption and signature schemes that align well with the use of the Groth-Sahai framework [18,20] to verify the relation R_{ZK}. We achieve such a combination by using the (pairing-based) structure-preserving signature (SPS) scheme from Kiltz et al. [23] in combination with the (pairing-based) inner-product PE scheme from Okamoto et al. [29] that yields pairing product equations to verify relation R_{ZK}.

In a nutshell, pairing groups are represented as a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of prime order q, g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Finally, $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ is an efficiently computable non-degenerate bilinear map and $g_T := e(g_1, g_2)$ is a generator of the target group. Groth-Sahai proofs implement a NIZK for a collection of product pairing

equations of the form

$$\prod_{i=1}^{s} e(x_i, A_i) \cdot \prod_{i=1}^{s'} e(B_i, y_i) \cdot \prod_{i=1}^{s'} \prod_{j=1}^{s} e(x_i, y_i)^{\gamma_{i,j}} = t_T$$

where $A_i \in \mathbb{G}_1$, $B_i \in \mathbb{G}_2$, $t_T \in \mathbb{G}_T$ and $\gamma_{i,j} \in \mathbb{Z}_q$ are constants (and part of the statement to be proven), and $x_i \in \mathbb{G}_1$ as well as $y_i \in \mathbb{G}_2$ are the private witness variables (and s, s' are integers). A priori, GS proofs for product pairing equations are only witness-indistinguishable unless certain additional constraints are met [20]. But even if those conditions are not met, efficient transformations can turn GS NIWI into full NIZK proofs (with extractability for group elements) with low overhead as shown in [13] by creating an OR-Proof system (allowing a simulator to always find a witness) and using the controlled malleability of the GS proof systems. We refer to [14, Theorem 3.2 and Appendix B] for the full details. As mentioned above, we instantiate the paring-based primitives from [29] (encryption) and [23] (signature):

- In the PE scheme of [29], ciphertexts are represented as pairs $\mathsf{ct} = (c_1, c_2)$, where $c_2 \in \mathbb{G}_T$ is the blinded plaintext m and has the form $c_2 = m \cdot g^{\zeta}$ (for a random ζ chosen during encryption) and c_1 is an N-vector $c_1 = (A_1, \ldots A_N)$ with $A_i \in \mathbb{G}_1$ (for an integer parameter N of the scheme). The decryption key for functionality $f_{\boldsymbol{x}}$ is represented by an N-vector $\mathsf{sk}_{f_{\boldsymbol{x}}} = (k_1, \ldots, k_N)$ with $k_i \in \mathbb{G}_2$. The decryption operation is $m' \leftarrow c_2 / \prod_{i=1}^{N} e(A_i, k_i)$. Note that to turn the scheme into a predicate-only PE scheme, we can fix $m = \mathbb{I}_{G_T}$ and do not need the extra blinding of the ciphertext (fixing $\zeta := 1$) and hence the decryption operation satisfies the equation $c_2 = g_T = e(g_1, g_2) = \prod_{i=1}^{N} e(A_i, k_i)$.
- In the SPS scheme of [23], a signature string is a tuple $\sigma = (s_1, s_2, s_3, s_4)$ with $s_4 \in \mathbb{G}_2$, and $s_i \in \mathbb{G}_1^{1\times(k+1)}$, $i \in \{1, 2, 3\}$ for an integer parameter k. The public key of this system consists of four matrices $\mathbf{M}_i$, where $\mathbf{M}_1, \mathbf{M}_2 \in \mathbb{G}_2^{k+1\times k}$, $\mathbf{M}_3 \in \mathbb{G}_2^{n+1\times k}$, and $\mathbf{M}_4 \in \mathbb{G}_2^{k+1\times k}$ (where n is the parameter specifying the message length). Verifying a signature σ with respect to this public key amounts to the following collection of $2k + 1$ pairing product equations, where a message $x \in \mathbb{G}_1^n$ is encoded as $m := (g_1, x_1, \ldots, x_n)$: For each $j \in [k]$ we check that

$$\prod_{i=1}^{k+1} e((s_1)_i, (\mathbf{M}_4)_{j,i}) = \prod_{i=1}^{k+1} e((m)_i, (\mathbf{M}_3)_{j,i}) \cdot \prod_{i=1}^{k+1} e((s_2)_i, (\mathbf{M}_1)_{j,i}) \cdot \prod_{i=1}^{k+1} e((s_3)_i, (\mathbf{M}_2)_{j,i})$$

  holds, as well as $e((s_2)_j, s_4) = e((s_3)_j, g_2)$ is satisfied for each $j \in [k + 1]$.

Therefore, the relation in Fig. 4 can be expressed as proving a satisfying assignment of the above pairing product equations, where the (private) decryption

key and the private signature are the private witness variables of the above equations, whereas the ciphertext and public keys can be treated as the constants (and hence part of the statement).

Instantiating logical formulas. By applying the techniques of [22] in our setting, we can implement various policies expressed as logical formulas. While all previous techniques are applicable to our setting, we only dive into simple reductions for completeness, as the core principle is the same for any technique mapping a logical formula to inner-products or hidden-vector functionalities.

IPE and threshold clauses. Assume a finite list of variables P_i, $i = 1 \dots q$, where each variable can take on values p_i from a finite set $\mathcal{P}$. Assume a policy F is expressed as a combination of sender and receiver properties. We assume that the policy is expressed as a list of requirements, each requirement being a clause, and where one requires that exactly d out of k of the requirements (clauses) must be satisfied (e.g., $d = 1$ as in our introductory example).

That is, we have a set of clauses $\{K_i\}_{i\in[k]}$, each with n_i sender properties and m_i receiver properties of the form

$$K_i = (P^{(s)}_{idx(i,1)} = p_{i,1} \wedge \dots \wedge P^{(s)}_{idx(i,n_i)} = p_{i,n_i} \wedge P^{(r)}_{idx(i,n_i+1)} = p_{i,n_i+1} \wedge \dots \wedge P^{(r)}_{idx(i,n_i+m_i)} = p_{i,n_i+m_i}),$$

which we call a conjunctive clause. Here, $P^{(s)}_{idx(i,j)}$ resp. $P^{(r)}_{idx(i,j)}$ denote variables P_h indexed via an indexing function $h = idx(i,j)$ (which is induced by such a finite policy). Note that the variables constrain the sender (superscript (s)) and the receiver (superscript (r)).

Our goal is to map the policy F to the functionality class $\mathcal{F}^{\mathsf{IP}}_{k+1}$. In particular, we must show how the authority performs the mapping $(f_{\boldsymbol{x}}, \boldsymbol{x}) \leftarrow \mathsf{SubPol}(F, (\overline{x}_1, \dots, \overline{x}_n))$ in the scheme of Fig. 3, where $\overline{x}_1, \dots, \overline{x}_n$ is the assignment of attributes to each P_i of a user Alice (note that we omit treating null values for simplicity). The authority performs the following computation:

1. The authority precomputes which clauses Alice cannot satisfy anymore, and which ones she potentially can satisfy with a matching receiver. The authority defines for all $i \in [k]$:

$$X_i := \begin{cases} 1 & \text{if } \bigwedge_{j=1}^{n_i} (\overline{x}_{idx(i,j)} = p_{i,j}) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

   The first part of the output of the subpolicy algorithm SubPol is $f_{\boldsymbol{x}}(\cdot) := F^{\mathsf{IP}}_{N,k+1}((X_1, \dots, X_k, 1), \cdot)$, where we assume $N > k$.
2. The authority precomputes which clauses Alice cannot satisfy if she is the receiver, and which ones she potentially can satisfy with a matching sender.

The authority defines:

$$Y_i := \begin{cases} 1 & \text{if } \bigwedge_{j=1}^{m_i} (\overline{x}_{idx(i,n_i+j)} = p_{i,n_i+j}) = 1, \\ 0 & \text{otherwise,} \end{cases}$$

for all $i \in [k]$. The second part of the output of the subpolicy algorithm SubPol is $\boldsymbol{x} := (Y_1, \ldots, Y_k, -d)$.

We observe that if a sender obtains a secret key generated based on the vector $(X_1, \ldots, X_k, 1)$ and signs (as shown in Fig. 3) a message for a receiver public key that contains the ciphertext generated based on vector $(Y_1, \ldots, Y_k, -d)$ as shown above, we have

$$\begin{aligned} & \langle (X_1, \ldots, X_k, 1), (Y_1, \ldots, Y_k, -d) \rangle = 0 \\ \iff & \langle (X_1, \ldots, X_k), (Y_1, \ldots Y_k) \rangle = d \end{aligned}$$

because $N > k$ (which is assumed to avoid wraparound complications). Since each of the products $X_i \cdot Y_i$ signals the joint fulfillment of the original clause K_i (thanks to the precomputation step), this means that exactly d clauses are jointly satisfied, which corresponds to the policy F.

We note that if the policy F has disjunctive clauses instead, that is for each $i \in [k]$

$$\begin{aligned} K_i = (P^{(s)}_{idx(i,1)} = p_{i,1} \vee \cdots \vee P^{(s)}_{idx(i,n_i)} = p_{i,n_i} \vee P^{(r)}_{idx(i,n_i+1)} \\ = p_{i,n_i+1} \vee \cdots \vee P^{(r)}_{idx(i,n_i+m_i)} = p_{i,n_i+m_i}), \end{aligned}$$

(where for $d = k$ we obtain CNF formulas) an analogous reasoning yields that the reduction to inner products for dimension $2k + 1$ can be achieved by having the authority follow the above steps but define for all $i \in [k]$, $X_{2i-1} := 1$ and

$$X_{2i} := \begin{cases} 1 & \text{if } \bigvee_{j=1}^{n_i} (\overline{x}_{idx(i,j)} = p_{i,j}) = 1, \\ 0 & \text{otherwise,} \end{cases}$$

as well as for all $i \in [k]$

$$Y_{2i-1} := \begin{cases} 1 & \text{if } \bigvee_{j=1}^{m_i} (\overline{x}_{idx(i,n_i+j)} = p_{i,n_i+j}) = 1, \\ 0 & \text{otherwise,} \end{cases}$$

$$Y_{2i} := \begin{cases} 0 & \text{if } \bigvee_{j=1}^{m_i} (\overline{x}_{idx(i,n_i+j)} = p_{i,n_i+j}), \\ 1 & \text{otherwise.} \end{cases}$$

The authority finally computes $f_{\boldsymbol{x}}(\cdot) := F^{\mathrm{IP}}_{N,2k+1}((X_1,\ldots,X_{2k},1),\cdot)$ and $\boldsymbol{x} := (Y_1,\ldots,Y_{2k},-d)$ (and generates the associated keys and ciphertext as prescribed in Fig. 3). The above is seen to represent the policy F by observing that each clause i is represented by two variables such that the sum $X_{2i-1}\cdot Y_{2i-1} + X_{2i}\cdot Y_{2i}$ equals 0 if no party satisfies the clause, and 1 in any other case.

HVE and CNF formulas. HVE opens up the space for many policies and is itself realizable from the inner product functionality [8, 22]. For example, for CNF formulas, i.e., as above with $d = k$, where k is the number of disjunctive clauses, the reduction to HVE for dimension k is straightforward: The authority defines

$$X_i = \begin{cases} * & \text{if } \bigvee_{j=1}^{n_i} (\overline{x}_{idx(i,j)} = p_{i,j}) = 1, \\ 1 & \text{otherwise.} \end{cases}$$

and

$$Y_i = \begin{cases} 1 & \text{if } \bigvee_{j=1}^{m_i} (\overline{x}_{idx(i,n_i+j)} = p_{i,n_i+j}) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

The the authority computes $f_{\boldsymbol{x}}(\cdot) := F^{\mathrm{HV}}_{k}((X_1,\ldots,X_k),\cdot)$ and $\boldsymbol{x} := (Y_1,\ldots,Y_k)$ and generates the associated keys and ciphertext as prescribed in Fig. 3.

This accurately represents the CNF policy F: A sender can only decrypt the ciphertext in the public key of a receiver if for each clause i, either the sender already satisfies that clause and thus the resulting vector has the wildcard symbol $*$ at this position, or the receiver has a satisfying assignment and hence its vector must be equal to 1 at this position to match the sender's value.

5 Universal Composability and SIM-Based PCS

Simulation-based security has the advantage that, instead of arguing and excluding trivial attacks, we follow the real/ideal world paradigm, where in the ideal-world, the leakage to the simulator and the unforgeability properties are captured in an explicit fashion.

The ideal PCS functionality. In this section, we cast policy compliant signature as an enhanced signature functionality following [2, 11] that incorporates all of our declared goals for this primitive. We give the description in Section 5. The difference to a standard signature functionality are at a high-level the following:

- There is a distinct trusted party, denoted M that is responsible for the setup. M is responsible to generate the signing keys for parties with respect to the attributes they possess. Note that at this level of abstraction, we do not discuss *how* the authority decides to assign an attribute to a party. This will be managed by the higher-level protocols. The attributes of honest parties do not leak to the adversary, which captures that the obtained public key

does not leak any attributes. However, the adversary learns by definition of the signature algorithm, whether the corrupted parties are allowed to send messages to the new honest parties.
- On signing operations, only valid signatures are recorded. That is, if party P_i with attributes x_{P_i} signs a message m for party P_j with attributes x_{P_j}, then the record $(m, \sigma, v_M, v_{P_i}, v_{P_j}, 1)$ is only stored if $F(x_{P_i} x_{P_j}) = 1$, where v_M denotes the public parameters and v_{P_i}, v_{P_j} are the unique public keys associated with parties P_i and P_j, respectively.
- On verification queries of the form $(\text{VERIFY}, \text{sid}, m, \sigma, v'_M, v'_A, v'_B)$, the functionality ensures aside of completeness and unforgeability w.r.t. honest signers also that no valid signature can be generated for any combination of v'_A, v'_B unless the public keys are associated to attributes x'_A and x'_B such that $F(x'_A, x'_B) = 0$.
- On top of unforgeability, privacy guarantees that the adversary learns at most the policy evaluation $F(x_i, x_j)$ (associated with the respective keys) for every signing query. For corrupted parties, the adversary learns their attributes $\tilde{x}$ (since it learns all inputs and outputs by that party upon corruption by default) as well as all evaluations $F(\tilde{x}, x_j)$.

Functionality $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$

The functionality interacts with an arbitrary party set $\mathcal{P} := \{P_1, \ldots, P_n\}$ and adversary $\mathcal{S}$. The functionality is parameterized by a distinct identity $M \notin \mathcal{P}$ of the credential manager and the class of supported policies $\mathcal{F}$.
Initialize $F \leftarrow \perp$, $x_{P_i}, v_{P_i} \leftarrow \perp$ for all $P_i \in \mathcal{P} \backslash \{M\}$ and $v_M \leftarrow \perp$. The functionality maintains the initialized party set $\mathcal{I} := \{P_i \in \mathcal{P} \mid v_{P_i} \neq \perp\}$ (and we omit the explicit inclusion of parties for simplicity).

Policy Initialization. Upon input $(\text{POLICY-GEN}, \text{sid}, F)$ from party M do the following: if $v_M \neq \perp$ or $F \notin \mathcal{F}$, ignore the request; otherwise, provide $(\text{POLICY-GEN}, \text{sid}, F)$ to $\mathcal{S}$. Upon receiving $(\text{POLICY-GEN}, \text{sid}, v)$ from $\mathcal{S}$, output $(\text{POLICY-GEN}, \text{sid}, v)$ to M and set $v_M \leftarrow v$.

Key Generation. Upon input $(\text{KEY-GEN}, \text{sid}, P, x)$ from party M, where $P \in \mathcal{P} \backslash \mathcal{I}$, do the following: ignore the request if $v_M = \perp$; otherwise define $x_{P_i} \leftarrow x$ and compute:

1. Provide the leakage information $\{(\hat{P}, P_i) \mapsto F(x_{\hat{P}}, x_{P_i}) \mid \text{for all corrupted } \hat{P} \in \mathcal{I}\}$ to $\mathcal{S}$.
2. Provide $(\text{KEY-GEN}, \text{sid}, P_i)$ to $\mathcal{S}$. Upon receiving $(\text{VERIFICATION-KEY}, \text{sid}, P_i, v)$ from $\mathcal{S}$, verify that for all $P \in \mathcal{I}$ $v_P \neq v$, and if this is the case, set $v_{P_i} \leftarrow v$ and output $(\text{VERIFICATION-KEY}, \text{sid}, x, v)$ to P_i. If v is not unique, ignore the input from $\mathcal{S}$.

Signing. On input $(\text{SIGN}, \text{sid}, m, v)$ from party $P \in \mathcal{I}$:

- If $v = v_{P_j}$ for some $P_j \in \mathcal{I}$ and $F(x_P, x_{P_j}) = 1$ then provide $(\text{SIGN}, \text{sid}, m, P, v, 1)$ to $\mathcal{S}$. Upon receiving $(\text{SIGNATURE}, \text{sid}, m, P, v, \sigma)$ from $\mathcal{S}$, verify that no entry $(m, \sigma, v_M, v_P, v_{P_j}, 0)$ is stored. If it is, then output an

error message to P and halt. Else, output $(\text{SIGNATURE}, \text{sid}, m, v, \sigma)$ to P, and store the entry $(m, \sigma, v_M, v_P, v_{P_j}, 1)$.
– In any other case, provide $(\text{SIGN}, \text{sid}, m, P, v, 0)$ to $\mathcal{S}$ and when receiving $(\text{SIGNATURE}, \text{sid}, m, s)$ from $\mathcal{S}$, output $(\text{SIGNATURE}, \text{sid}, m, v, s)$ to P. *(This case guarantees that such messages are not considered as signed.)*

Verification. Upon input $(\text{VERIFY}, \text{sid}, m, \sigma, v'_M, v'_A, v'_B)$ from any party P, hand $(\text{VERIFY}, \text{sid}, m, \sigma, v'_M, v'_A, v'_B)$ to $\mathcal{S}$. Upon receiving $(\text{VERIFIED}, \text{sid}, m, v'_M, v'_A, v'_B, \phi)$ from $\mathcal{S}$ do:

1. If $v'_M = v_M$, $v'_A = v_{P_i}$, $v'_B = v_{P_j}$ for some $P_i, P_j \in \mathcal{I}$ and the entry $(m, \sigma, v_M, v_{P_i}, v_{P_j}, 1)$ is recorded, then set $f = 1$. *(Condition 1 guarantees completeness: If the verification keys are the registered ones and σ is a legitimately generated signature for m, then the verification succeeds.)*
2. Else, if $v'_M = v_M$, $v'_A = v_{P_i}$, $v'_B = v_{P_j}$ for some $P_i, P_j \in \mathcal{I}$ and P_i is not corrupted and no entry $(m, \sigma, v_M, v_{P_i}, v_{P_j}, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, v_M, v_{P_i}, v_{P_j}, 0)$. *(Condition 2 guarantees unforgeability: For any combination of generated public keys, the signer is not corrupted, and never signed m, then the verification fails.)*
3. Else, if $v'_M = v_M$, $v'_A = v_{P_i}$, $v'_B = v_{P_j}$ for some $P_i, P_j \in \mathcal{I}$ and no entry $(m, \sigma, v_M, v_{P_i}, v_{P_j}, 1)$ for any σ' is recorded, then set $f = 0$ if $F(x_{P_i}, x_{P_j}) = 0$, and otherwise set $f \leftarrow \phi$. Record the entry $(m, \sigma, v_M, v_{P_i}, v_{P_j}, f)$. *(Condition 3 guarantees policy compliance of dishonest signers: For any combination of generated public keys, even if everyone is corrupted the verification must fail if the policy is not satisfied.)*
4. Else, if $v'_M = v_M$ but we have that $\forall P_i \in \mathcal{I} : v'_A \neq v_{P_i}$ or $\forall P_i \in \mathcal{I} : v'_B \neq v_{P_i}$, then set $f \leftarrow 0$ and record the entry $(m, \sigma, v_M, v'_A, v_{B'}, 0)$. *(Condition 4 ensures that no valid signatures can exist w.r.t. public keys not issued by the credential manager.)*
5. Else, if there is an entry (m, σ, v', f') stored, then let $f = f'$. *(Condition 5 guarantees consistency: All verification requests with identical parameters will result in the same answer.)*
6. Else, let $f = \phi$ and record the entry $(m, \sigma, v'_M, v'_A, v'_B, \phi)$. *(If no condition applies, then let the adversary decide.)*
7. Finally, output $(\text{VERIFIED}, \text{sid}, m, f)$ to P.

Corruption Mode. The party M is not corruptible. For all other parties P_i, the functionality supports the standard corruption mode [12], that is, upon input $(\text{CORRUPT}, P_i)$ on the backdoor tape, send all previous inputs to $\mathcal{S}$ and hand over the control of P_i's input and output tapes to $\mathcal{S}$ and providing the adversary all capabilities that an honest party has. Additionally, whenever a party P_i gets corrupted, provide the leakage information $\{(P_i, P_j) \mapsto F(x_{P_i}, x_{P_j}) \mid P_j \in \mathcal{I}\}$.

Blueprint usage of the scheme in UC. As with signatures [2,11], a PCS scheme $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ can be mapped in a straightforward way to a UC protocol tailored to realize the low-level functionality $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ (low level in the sense that it exports the interface without much abstraction). The main

difference to an ordinary signature scheme is the presence of a trusted party assisting in the key generation step. That is, we have a trusted (credential) manager M incorruptible by definition[6], where we assume secure point-to-point channels between M and each P_i. The protocol π_M^{PCS} can be specified as follows:

- **Party M:**
  - On input $(\textsc{policy-gen}, \mathrm{sid}, F)$, run $\mathsf{Setup}(1^\lambda, F)$ and generate the output $(\textsc{policy-gen}, \mathrm{sid}, \mathsf{mpk})$. Store msk internally.
  - On input $(\textsc{key-gen}, \mathrm{sid}, P, x)$, run $\mathsf{KeyGen}(\mathsf{msk}, x)$ and send the message $(x, (\mathsf{pk}, \mathsf{sk}))$ to party P over a secure channel.
- **Party P_i:**
  - Upon receiving (for the first time) the message $(x, (\mathsf{pk}, \mathsf{sk}))$ from M on the secure channel, store it internally and output $(\textsc{verification-key}, \mathrm{sid}, x, \mathsf{pk})$. If the party has initialized its public key already, messages from M are ignored.
  - On input $(\textsc{sign}, \mathrm{sid}, m, v)$, if this party has already a secret key sk, then execute $\sigma \leftarrow \mathsf{Sign}(v, \mathsf{sk}, m)$ and return $(\textsc{signature}, \mathrm{sid}, m, v, \sigma)$.
  - On input $(\textsc{verify}, \mathrm{sid}, m, \sigma, v'_M, v'_A, v'_B)$, return the output $(\textsc{verified}, \mathrm{sid}, m, \mathsf{Verify}(v'_M, v'_A, v'_B, m, \sigma))$.

With this composable understanding in mind, we now set out to establish a concise and simpler SIM-based PCS notion in the spirit of [7, 24, 27] that implies the UC realization of the ideal PCS functionality, which we show formally in Theorem 5.2. Looking ahead, the proof of Theorem 5.2 reveals that all the ideal unforgeability properties (Conditions 2, 3, and 4) of $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ follow from the game-based unforgeability notion defined in Definition 3.2, which is thereby validated to capture what we intended to model.

5.1 Simulation-Based Attribute Hiding

Our starting point is the already established game-based notion, where the adversary gets access to a variety of oracles, as defined in Section 3.1. Following [7, 24, 27], we consider a simulator $\mathcal{S} = (\mathcal{S}_{\mathrm{Setup}}, \mathcal{S}_{\mathrm{KG}}, \mathcal{S}_{\mathrm{Cor}}, \mathcal{S}_{\mathrm{Sgn}})$, where $\mathcal{S}_{\mathrm{Setup}}$ simulates Setup and $\mathcal{S}_{\mathrm{KG}}$, $\mathcal{S}_{\mathrm{Cor}}$, and $\mathcal{S}_{\mathrm{Sgn}}$ simulate the oracles $\mathsf{QKeyGen}$, QCor, and QSign, respectively. These simulator algorithms have a shared state and in addition to the inputs to the oracles, get a *leakage set* $\mathcal{L}$. The set $\mathcal{L}$ is initially empty and gets augmented during the experiment analogous to how the simulator in the UC functionality obtains information.

Definition 5.1 (SIM-Based AH). *Let* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a PCS scheme as defined in Definition 3.1. We define the experiments* $\mathrm{Real}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ *and* $\mathrm{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S})$ *for a PPT adversary* $\mathcal{A}$ *and a PPT simulator* $\mathcal{S} = (\mathcal{S}_{\mathrm{Setup}}, \mathcal{S}_{\mathrm{KG}}, \mathcal{S}_{\mathrm{Cor}}, \mathcal{S}_{\mathrm{Sgn}})$ *in Fig. 6. In the real world, the adversary*

[6] Formally, the property of such trusted third parties to be incorruptible is modeled by instructing its protocol machine to ignore the corruption request on the backdoor tape.

has access to oracles as defined in Section 3.1. The simulator algorithms have a shared state s, which is modelled as giving them s as input, and allowing all of them to update the state s. In the ideal experiment, the initially empty sets $\mathcal{IQK}$ and $\mathcal{IQC}$ are maintained. Furthermore, all but $\mathcal{S}_{\text{Setup}}$ get as an additional input the leakage set $\mathcal{L}$, which is initially empty. The sets are updated according to the following rules:

- *When $\mathcal{A}$ queries the key generation oracle using x_j, the following gets added to $\mathcal{L}$ (before $\mathcal{S}_{\text{KG}}$ is invoked):*

$$\{(i,j) \mapsto F(x_i, x_j) \mid (i, \mathsf{pk}_i, x_i) \in \mathcal{IQC}\}.$$

  After the simulator $\mathcal{S}_{\text{KG}}$ has been invoked, (j, pk_j, x_j) is added to $\mathcal{QK}$, where pk_j is the output of $\mathcal{S}_{\text{KG}}$.
- *When $\mathcal{A}$ makes a corruption query i with $(i, \mathsf{pk}_i, x_i) \in \mathcal{QK}$, then the following gets added to $\mathcal{L}$ (before $\mathcal{S}_{\text{Cor}}$ is invoked):*

$$\big(x_i, \{(i,j) \mapsto F(x_i, x_j) \mid (j, \mathsf{pk}_j, x_j) \in \mathcal{IQK}\}\big).$$

  Additionally, $(i, \mathsf{pk}_i, x_i) \in \mathcal{IQK}$ is also added to $\mathcal{IQC}$.
- *When $\mathcal{A}$ makes a signing query (i, pk_R, m), the following gets added to $\mathcal{L}$:*

$$\{(i,j) \mapsto F(x_i, x_j) \mid (i, \mathsf{pk}_i, x_i), (j, \mathsf{pk}_R, x_j) \in \mathcal{IQK}\}.$$

  This models that adversaries learn whether a pair of keys satisfy the policy by observing a signature for these keys.

The advantage of a PPT adversary $\mathcal{A}$ in the experiment is defined as:

$$\mathsf{Adv}^{\text{Sim}}_{\mathsf{PCS},\mathcal{A},\mathcal{S}}(\lambda) = |\Pr[\text{Real}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\text{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S}) = 1]|.$$

A PCS scheme PCS is simulation attribute hiding, if for any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$, such that $\mathsf{Adv}^{\text{Sim}}_{\mathsf{PCS},\mathcal{A},\mathcal{S}}(\lambda) \leq \text{negl}(\lambda)$, where $\text{negl}(\cdot)$ is a negligible function.

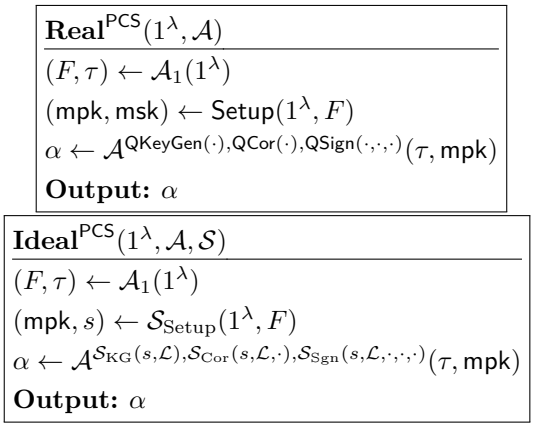

Fig. 6: Real and ideal experiments for the simulation-based attribute hiding definition for the scheme PCS. Both experiments interact with an adversary $\mathcal{A}$. The ideal experiment additionally interacts with a simulator $\mathcal{S} = (\mathcal{S}_{\text{Setup}}, \mathcal{S}_{\text{KG}}, \mathcal{S}_{\text{Cor}}, \mathcal{S}_{\text{Sgn}})$. The simulator gets the initially empty leakage set $\mathcal{L}$, which grows during the experiment as described in Definition 5.1.

We conclude with the following theorem:

Theorem 5.2. *Protocol π_M^{PCS} securely realizes $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ if PCS is existentially unforgeable (Definition 3.2) and simulation-based attribute hiding (Definition 5.1).*

Proof. We prove the theorem in two main steps. First we assume a hybrid world with a functionality like $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ but which does not protect the privacy of any party's attributes, but only enforces the ideal unforgeability guarantees. We show that there is a UC simulator $\mathcal{S}_{uc}$ that can simulate the real-world perfectly unless the environment (together with the dummy adversary) provoke an event that implies a forgery of the PCS scheme as captured by game $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}$ in Fig. 1. The second step of the proof is to switch to the true ideal world with $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$. We re-design the previous simulator to obtain $\mathcal{S}'_{uc}$ that uses the assumed simulator $\mathcal{S}_{pcs} = (\mathcal{S}_{\text{Setup}}, \mathcal{S}_{\text{KG}}, \mathcal{S}_{\text{Cor}}, \mathcal{S}_{\text{Sgn}})$ required by Definition 5.1. Any environment that notices this switch to $\mathcal{S}'_{uc}$ can be used to distinguish $\mathbf{Real}^{\mathsf{PCS}}$ and $\mathbf{Ideal}^{\mathsf{PCS}}$.

In more detail, we have the following hybrid worlds:

Hybrid H_0: This is the real-world process with protocol π_M^{PCS}.

Hybrid H_1: Here we assume an "ideal functionality" Func_{hyb} that acts like $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ but with the following difference:

- On input $(\text{KEY-GEN}, \text{sid}, P, x)$, behave as $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ does but additionally output x to the adversary.

Designing a simulator for this world follows the pattern of the signature simulator of [11] with additional setup, that is. We define the simulator $\mathcal{S}_{uc}$:

- On input $(\text{POLICY-GEN}, \text{sid}, F)$ from Func_{hyb}, execute $\mathsf{Setup}(1^\lambda, F)$ and then return to the functionality $(\text{POLICY-GEN}, \text{sid}, \mathsf{mpk})$. Store msk for future use.
- On input $(\text{KEY-GEN}, \text{sid}, P_i)$ alongside the leakage set $\{(\hat{P}, P_i) \mapsto F(x_{\hat{P}}, x_{P_i}) \,|\, \text{for all corrupted } \hat{P} \in \mathcal{I}\}$, *and the additional leakage information* x specific to Func_{hyb}, the simulator executes $\mathsf{KeyGen}(\mathsf{msk}, x)$, stores the obtained key-pair $(\mathsf{pk}, \mathsf{sk})$ as $(P_i, \mathsf{pk}, \mathsf{sk})$ for future use. Provide $(\text{VERIFICATION-KEY}, \text{sid}, P_i, \mathsf{pk})$ to the functionality.
- On input $(\text{SIGN}, \text{sid}, m, P, v, b)$ from Func_{hyb}, obtain the record $(P, \mathsf{pk}, \mathsf{sk})$ and execute $\sigma \leftarrow \mathsf{Sign}(v, \mathsf{sk}, m)$ (give up activation if this party has not yet obtained its key). Return to the functionality $(\text{SIGNATURE}, \text{sid}, m, P, v, \sigma)$.
- On input $(\text{VERIFY}, \text{sid}, m, \sigma, v'_M, v'_A, v'_B)$ from Func_{hyb}, let $\phi \leftarrow \mathsf{Verify}(v'_M, v'_A, v'_B, m, \sigma)$ and return $(\text{VERIFIED}, \text{sid}, m, v'_M, v'_A, v'_B, \phi)$ to the functionality.
- On a corruption request for party P_i, $\mathcal{S}_{uc}$ corrupts P_i in the ideal functionality (and formally also obtains leakage set $\{(P_i, P_j) \mapsto F(x_{P_i}, x_{P_j}) \,|\, P_j \in \mathcal{I}\}$ that is not needed here since the simulator has full knowledge of attributes), checks for a previously stored record $(P_i, \mathsf{pk}, \mathsf{sk})$ and if such a record exists returns sk. (And from now onward, the simulator acts as relay between environment and functionality.)

Hybrid H_2: This hybrid is the ideal functionality (i.e., the ideal protocol for) $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ together with simulator $\mathcal{S}'_{uc}$. We define $\mathcal{S}'_{uc}$ by stating what the difference is compared to $\mathcal{S}_{uc}$. This will be handy when arguing about the indistinguishability of this and the previous hybrid.

- On input $(\text{POLICY-GEN}, \text{sid}, F)$ from $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$, simulator $\mathcal{S}'_{uc}$ executes $(\mathsf{mpk}, s) \leftarrow \mathcal{S}_{\text{Setup}}(1^\lambda, F)$ (instead of Setup) and stores s for future use (instead of msk). Initialize the leakage set $\mathcal{L} \leftarrow \emptyset$. The interaction with the functionality is just like $\mathcal{S}_{uc}$.
- On input $(\text{KEY-GEN}, \text{sid}, P_i)$ alongside the leakage set $L = \{(\hat{P}, P_i) \mapsto F(x_{\hat{P}}, x_{P_i}) \,|\, \text{for all corrupted } \hat{P} \in \mathcal{I}\}$—*but without the additional leakage* x *from above*–from $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$, $\mathcal{S}'_{uc}$ computes $\mathcal{L} \leftarrow \mathcal{L} \cup L$ and executes $\mathcal{S}_{\text{KG}}(s, \mathcal{L})$ to obtain pk and an updated state s. The simulator stores the tuple $(P_i, \mathsf{pk}, \perp)$ (no secret key is stored). The remaining interaction with the functionality is identical to $\mathcal{S}_{uc}$.
- On input $(\text{SIGN}, \text{sid}, m, P, v, b)$ from $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$, the simulator updates the leakage set $\mathcal{L}$ only if there is an entry $(P', v, \cdot)$ by adding the tuple (P, P', b). Next, retrieve a previously stored record $(P, \mathsf{pk}, \perp)$ and generate the signature $\sigma \leftarrow \mathcal{S}_{\text{Sgn}}(s, \mathcal{L}, \mathsf{pk}, v)$ (which also updates the state s). The interaction between the simulator and the functionality is the same as in $\mathcal{S}_{uc}$.
- On input $(\text{VERIFY}, \text{sid}, m, \sigma, v'_M, v'_A, v'_B)$ from $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$, the simulator behaves identically to $\mathcal{S}_{uc}$.
- On a corruption request for party P_i, $\mathcal{S}'_{uc}$ corrupts P_i in $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$, and includes the additional leakage information $L = \{(P_i, P_j) \mapsto F(x_{P_i}, x_{P_j}) \,|\, P_j \in \mathcal{I}\}$ by computing $\mathcal{L} \leftarrow \mathcal{L} \cup L$. Next, it retrieves the record $(P_i, \mathsf{pk}, \cdot)$ and

if such a record exists returns $\mathsf{sk} \leftarrow \mathcal{S}_{\mathrm{Cor}}(s, L, \mathsf{pk})$ (which also updates s) and returns sk. (And from now, the simulator acts as relay between environment and functionality.)

In the full version, we state and prove two claims that show the indistinguishability of hybrid H_0 and H_1 and the indistinguishability of H_1 and H_2. Combining both claims yields that for any UC environment (and without loss of generality in the dummy adversary model, see [10]) the (real) protocol execution of π_M^{PCS} is indistinguishable from the ideal protocol execution with functionality $\mathsf{Func}_{\mathsf{PolSig}}^{M,\mathcal{F}}$ and ideal adversary (i.e., simulator) $\mathcal{S}'_{uc}$. The theorem follows. □

5.2 On the SIM-Based Security of our Generic Scheme

If we assume that the underlying predicate-only predicate encryption scheme of our construction in Fig. 3 satisfies the strong simulation-based PE security notion as defined in the full version, then the generic scheme achieves the simulation-based and therefore the composable notion of PCS. We note that the requirement in the simulation-based PE security is the adaptive (and thus stronger) version of what is proven so far in the literature, such as in [17, 29]. We leave it as an interesting open problem to realize PE schemes that fulfill the stronger (adaptive) simulation-based security notion based on reasonable assumptions. We note that such schemes require idealized models such as proofs in the bilinear generic group model [24] or the random oracle model.

Theorem 5.3. *Let* $\mathsf{PE} = (\mathsf{PE.Setup}, \mathsf{PE.KeyGen}, \mathsf{PE.Enc}, \mathsf{PE.Dec})$ *be a simulation secure predicate encryption scheme, let further* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *be a NIZK proof system and let* $\mathsf{DS_{pub}} = (\mathsf{DS_{pub}.Setup}, \mathsf{DS_{pub}.Sign}, \mathsf{DS_{pub}.Verify})$ *be a strong unforgeable signature scheme, then there exists a simulator* $\mathcal{S}$ *such that the construction* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figure 3, is simulation private. Namely, for any PPT adversary* $\mathcal{A}$ *there exist PPT adversaries* $\mathcal{B}, \mathcal{B}'$ *and* $\mathcal{B}''$*, such that:*

$$\mathsf{Adv}^{\mathrm{Sim}}_{\mathsf{PCS},\mathcal{A},\mathcal{S}}(\lambda) \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{Sim}}_{\mathsf{PE},\mathcal{B}',\mathcal{S}'}(\lambda) + \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS_{pub}},\mathcal{B}''}(\lambda).$$

Proof. The simulator $\mathcal{S}$ for the proof of this theorem is described in the full version. Informally, the simulator $\mathcal{S}$ uses the simulators of the predicate encryption scheme to generate the keys. For answering signature queries, the simulator $\mathcal{S}$ additionally receives the policy evaluation of the associated attributes of the keys that are used for the signature query. Since $\mathcal{S}$ knows if the statement is part of the language of the NIZK system, it can use the NIZK simulator to generate a valid proof for the statement relying on the zero-knowledge property of the NIZK system.

As in the proof of Theorem 4.3, we assume that the adversary $\mathcal{A}$ only queries the signing oracle using public keys that previously have been output by one of the key oracles or has been a reply to the challenge query. The argument here is the same as in the proof of Theorem 4.3, which results in the summand $\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS_{pub}},\mathcal{B}''}(\lambda)$ of the bound in the theorem.

To show that the ideal world with the simulator $\mathcal{S}$ is indistinguishable from the real world, we use a hybrid argument where the hybrids are formally defined in the full version. Note that H_0 corresponds to the real world $\text{Real}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ and H_2 to the ideal world $\text{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S})$. This results in:

$$\mathsf{Adv}^{\text{Sim}}_{\mathsf{PCS},\mathcal{A},\mathcal{S}}(\lambda) = |\Pr[\mathsf{H}_0(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{H}_2(\lambda, \mathcal{A}) = 1]|.$$

We describe the different games in more detail:

Hybrid H_1: In this hybrid, we change from an honestly generated CRS and honestly generated proofs to a simulated CRS and simulated proofs. The transition from H_0 to H_1 is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\Pr[\mathsf{H}_0(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{H}_1(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda).$$

Hybrid H_2: This hybrid is the $\text{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ world. In this hybrid, we change from honestly generated keys to simulated keys. The transition from H_1 to H_2 is justified by the simulation policy hiding property of PE. Namely, we can exhibit a PPT adversary $\mathcal{B}_1$ such that:

$$|\Pr[\mathsf{H}_1(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{H}_2(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\text{Sim}}_{\mathsf{PE},\mathcal{B}_1,\mathcal{S}'}(\lambda).$$

Putting everything together, we obtain the theorem. □

We present the proofs for the different transitions in the full version.

References

1. Ateniese, G., Francati, D., Nuñez, D., Venturi, D.: Match me if you can: Matchmaking encryption and its applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 701–731. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26951-7_24
2. Backes, M., Hofheinz, D.: How to break and repair a universally composable signature functionality. In: Zhang, K., Zheng, Y. (eds.) ISC 2004. LNCS, vol. 3225, pp. 61–72. Springer, Heidelberg (Sep 2004)
3. Badertscher, C., Matt, C., Maurer, U.: Strengthening access control encryption. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 502–532. Springer, Heidelberg (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_18
4. Bellare, M., Fuchsbauer, G.: Policy-based signatures. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 520–537. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0_30
5. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_4
6. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_30

7. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (Mar 2011). https://doi.org/10.1007/978-3-642-19571-6_16
8. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (Feb 2007). https://doi.org/10.1007/978-3-540-70936-7_29
9. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_7
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
11. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003), https://eprint.iacr.org/2003/239
12. Canetti, R.: Universally composable security. J. ACM **67**(5) (Sep 2020). https://doi.org/10.1145/3402457, https://doi.org/10.1145/3402457
13. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 281–300. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_18
14. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. Cryptology ePrint Archive, Report 2012/012 (2012), https://eprint.iacr.org/2012/012
15. Damgård, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., Siniscalchi, L.: Balancing privacy and accountability in blockchain identity management. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 552–576. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_23
16. Damgård, I., Haagh, H., Orlandi, C.: Access control encryption: Enforcing information flow with cryptography. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 547–576. Springer, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53644-5_21
17. Datta, P., Okamoto, T., Takashima, K.: Adaptively simulation-secure attribute-hiding predicate encryption. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 640–672. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_22
18. Escala, A., Groth, J.: Fine-tuning Groth-Sahai proofs. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 630–649. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0_36
19. Frikken, K.B., Li, J., Atallah, M.J.: Trust negotiation with hidden credentials, hidden policies, and policy cycles. In: NDSS 2006. The Internet Society (Feb 2006)
20. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_24
21. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 143–154. Springer, Heidelberg (May 1996). https://doi.org/10.1007/3-540-68339-9_13
22. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N.P. (ed.) EURO-

CRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_9
23. Kiltz, E., Pan, J., Wee, H.: Structure-preserving signatures from standard assumptions, revisited. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 275–295. Springer, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-48000-7_14
24. Kim, S., Lewi, K., Mandal, A., Montgomery, H., Roy, A., Wu, D.J.: Function-hiding inner product encryption is practical. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 544–562. Springer, Heidelberg (Sep 2018). https://doi.org/10.1007/978-3-319-98113-0_29
25. Li, N., Du, W., Boneh, D.: Oblivious signature-based envelope. In: Borowsky, E., Rajsbaum, S. (eds.) 22nd ACM PODC. pp. 182–189. ACM (Jul 2003). https://doi.org/10.1145/872035.872061
26. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 376–392. Springer, Heidelberg (Feb 2011). https://doi.org/10.1007/978-3-642-19074-2_24
27. Matt, C., Maurer, U.: A definitional framework for functional encryption. In: Fournet, C., Hicks, M. (eds.) CSF 2015 Computer Security Foundations Symposium. pp. 217–231. IEEE Computer Society Press (2015). https://doi.org/10.1109/CSF.2015.22
28. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. manuscript (2009), http://www.bitcoin.org/bitcoin.pdf
29. Okamoto, T., Takashima, K.: Adaptively attribute-hiding (hierarchical) inner product encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 591–608. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_35
30. Okamoto, T., Takashima, K.: Fully secure unbounded inner-product and attribute-based encryption. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 349–366. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4_22
31. Wee, H.: Attribute-hiding predicate encryption in bilinear groups, revisited. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 206–233. Springer, Heidelberg (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_8