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 extend 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.

Table of Contents

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 [MPR11] and policy-based signatures [BF14] 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.[1]

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 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, (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 the attribute hiding property of PCS ensures that signatures attest the validity without revealing the combination of attributes of the involved public keys.

Furthermore, in blockchain systems such as Bitcoin [Nak09], 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) signature (produced by the PCS scheme) 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 company-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 a signature and the address formats. The security of PCS makes it impossible for any sender to

[1] 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.

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 [CL01, DGK+21]. 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 [FLA06, LDB03]. 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 [LDB03], 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 [LDB03], what PCS bring 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

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

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 [KSW08]. 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 [BW07].

Composable PCS and SIM-based notion. Finally, we cast PCS as an ideal, enhanced signature functionality in the spirit of [Can03,BH04] 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 attribute-hiding 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 (e.g. in [DOT18]) so far.

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) [MPR11] 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 [BF14] 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. [JSI96]. 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. [AFNV19] 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 support. 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 Appendix A.

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 [AFNV19] are in the random oracle model.

Access control encryption. The notion of access control encryption (ACE) [DHO16, BMM17] 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 [BW07, KSW08]. In the work of Boneh and Waters [BW07], the authors construct a scheme that allows for comparison, subset and arbitrary conjunctive queries. In the succeeding work of Katz et al. [KSW08], 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 [BW07] and Katz et al. [KSW08], more works for the same functionality class have been proposed [OT12a, OT12b], as well as for the stronger notion of partially-hiding predicate encryption [Wee17, DOT18]. 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

2.1 Notation

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 $\lambda > \lambda_0$: $\mathrm{negl}(\lambda) < 1/p(\lambda)$. 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 black-box oracle access to $\mathcal{B}$, that is, the response for an oracle query q is $\mathcal{B}(q)$.

2.2 Digital Signatures

In this section, we recap the definition of digital signatures as well as existential unforgeability as introduced in [GMR88].

Definition 2.1 (Digital Signatures). *A digital signature scheme (DS) is a triple of PPT algorithms* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$*:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a security parameter λ and outputs a verification key* vk *and a signing key* sk*.*

$\mathsf{Sign}(\mathsf{sk}, m)$**:** *Takes as input the signing key* sk*, a message $m \in \mathcal{M}$ and outputs a signature σ.*

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$**:** *Takes as input the verification key* vk*, a message m and a signature σ, and outputs* 0 *or* 1*.*

A scheme DS *is* correct *if (for all $\lambda \in \mathbb{N}$), for all* vk *in the support of* $\mathsf{Setup}(1^\lambda)$ *and all $m \in \mathcal{M}$, we have*

$$\Pr[\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1.$$

Definition 2.2 (Existential Unforgeability of a Digital Signature Scheme). *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *be a DS scheme. We define the experiment* $\text{EUF-CMA}^{\mathsf{DS}}$ *in Fig. 1 with Q being the set containing the queries of $\mathcal{A}$ to the signing oracle* $\mathsf{Sign}(\mathsf{sk}, \cdot)$*. The advantage of an adversary $\mathcal{A}$ is defined by*

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

A Digital Signature scheme DS *is called* existentially unforgeable under adaptive chosen-message attacks (EUF-CMA secure) *if for any polynomial-time adversary $\mathcal{A}$ it holds that* $\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ *for a negligible function* $\mathrm{negl}(\cdot)$*.*

```
$\textbf{EUF-CMA}^{\mathsf{DS}}(1^\lambda, \mathcal{A})$
$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{vk})$
$\textbf{Output: } \mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 \land m \notin Q$
```

Fig. 1: Existentially Unforgeability Game of DS.

Definition 2.3 (Strong Unforgeability of a Digital Signature Scheme). *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *be a DS scheme. We define the experiment* EUF-CMA$^{\mathsf{DS}}$ *in Fig. 1 with* Q *being the set containing the queries of* $\mathcal{A}$ *to the signing oracle* $\mathsf{Sign}(\mathsf{sk}, \cdot)$ *and the corresponding answers to* $\mathcal{A}$*. The advantage of an adversary* $\mathcal{A}$ *is defined by*

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

A Digital Signature scheme DS *is called* strong unforgeable under adaptive chosen-message attacks (SUF-CMA secure) *if for any polynomial-time adversary* $\mathcal{A}$ *we have* $\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda)$ *for a negligible function* $\text{negl}(\cdot)$*.*

```
$\textbf{SUF-CMA}^{\mathsf{DS}}(1^\lambda, \mathcal{A})$
$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{vk})$
$\textbf{Output: } \mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 \land (m, \sigma) \notin Q$
```

Fig. 2: Strong Unforgeability Game of DS.

2.3 Non-interactive Zero-Knowledge Proofs

Now, we recapture the definition of non-interactive zero knowledge (NIZK) proofs [GMW87, For87, BGG+90].

Definition 2.4 (Non-Interactive Zero-Knowledge Proofs). *Let* R *be an NP Relation and consider the language* $L = \{x \mid \exists w \text{ with } (x, w) \in R\}$ *(where* x *is called a statement or instance). A non-interactive zero-knowledge proof (NIZK) for the relation* R *is a triple of PPT algorithms* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$*:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a security parameter* λ *and outputs the common reference string* CRS*.*
$\mathsf{Prove}(\mathsf{CRS}, x, w)$**:** *Takes as input the common reference string* CRS*, a statement* x *and a witness* w*, and outputs a proof* π*.*
$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$**:** *Takes as input the common reference string* CRS*, a statement* x *and a proof* π*, and outputs* 0 *or* 1*.*

A system NIZK *is complete, if (for all* $\lambda \in \mathbb{N}$*), for all* CRS *in the support of* $\mathsf{Setup}(1^\lambda)$ *and all statement-witness pairs in the relation* $(x, w) \in R$*,*

$$\Pr[\mathsf{Verify}(\mathsf{CRS}, x, \mathsf{Prove}(\mathsf{CRS}, x, w)) = 1] = 1.$$

Besides completeness, a NIZK system should also fulfill the notions of soundness and zero-knowledge, which we introduce in the following two definitions:

Definition 2.5 (Soundness). *Given a proof system* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for a relation* R *and the corresponding language* L*, we define the soundness advantage of an adversary* $\mathcal{A}$ *as the probability:*

$$\mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) := \Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(\mathsf{CRS}) : \mathsf{Verify}(\mathsf{CRS}, x, \pi) = 1 \wedge x \notin L].$$

A NIZK proof system is called perfectly sound if $\mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) = 0$ *for all algorithms* $\mathcal{A}$*, and computationally sound, if* $\mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ *for all PPT algorithms* $\mathcal{A}$*.*

$\mathbf{ZK}_0^{\mathsf{NIZK}}(1^\lambda, \mathcal{A}, \mathcal{S})$	$\mathbf{ZK}_1^{\mathsf{NIZK}}(1^\lambda, \mathcal{A}, \mathcal{S})$
$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$	$(\mathsf{CRS}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{Prove}(\mathsf{CRS},\cdot,\cdot)}(\mathsf{CRS})$	$\alpha \leftarrow \mathcal{A}^{\mathcal{S}'(\mathsf{CRS},\tau,\cdot,\cdot)}(\mathsf{CRS})$
Output: α	**Output:** α

Fig. 3: Zero-knowledge property of NIZK.

Definition 2.6 (Zero-Knowledge). *Let* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a NIZK proof system for a relation* R *and the corresponding language* L*,* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *a pair of algorithms (the simulator), with* $\mathcal{S}'(\mathsf{CRS}, \tau, x, w) = \mathcal{S}_2(\mathsf{CRS}, \tau, x)$ *for* $(x, w) \in R$*, and* $\mathcal{S}'(\mathsf{CRS}, \tau, x, w) = \mathsf{failure}$ *for* $(x, w) \notin R$*. For* $\beta \in \{0, 1\}$*, we define the experiment* $\mathrm{ZK}_\beta^{\mathsf{NIZK}}(1^\lambda, \mathcal{A})$ *in Fig. 3. The associated advantage of an adversary* $\mathcal{A}$ *is defined as*

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

A NIZK proof system NIZK *is called perfect zero-knowledge, with respect to a simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$*, if* $\mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) = 0$ *for all algorithms* $\mathcal{A}$*, and computationally zero-knowledge, if* $\mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) \leq \mathrm{negl}(\lambda)$ *for all PPT algorithms* $\mathcal{A}$*.*

Besides the notion of zero-knowledge and soundness, we also introduce the notion of extractability as in [CKLM12a].

Definition 2.7 (Extractability). *Let* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a NIZK proof system for a relation* R *and the corresponding language* L*,* $E = (E_1, E_2)$ *a pair of algorithms (the extractor). We define the extraction advantages of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathrm{CRS}}_{\mathsf{NIZK},\mathcal{A}} := |\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda); 1 \leftarrow \mathcal{A}(\mathsf{CRS})] - \Pr[\mathsf{CRS} \leftarrow E_1(1^\lambda); 1 \leftarrow \mathcal{A}(\mathsf{CRS})]|,$$

and

$$\begin{aligned}\mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) := \Pr[&\mathsf{CRS}_E \leftarrow E_1(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(\mathsf{CRS}_E) :\\ &\mathsf{Verify}(\mathsf{CRS}_E, x, \pi) = 1 \wedge R(x, E_2(\mathsf{CRS}_E, x, \pi)) = 0].\end{aligned}$$

A NIZK proof system NIZK *is called extractable, with respect to an extractor* $E = (E_1, E_2)$*, if* $\mathsf{Adv}^{\mathrm{CRS}}_{\mathsf{NIZK},\mathcal{A}} \leq \mathrm{negl}(\lambda)$ *and* $\mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$*. Additionally, we call an extractable non-interactive zero-knowledge proof a non-interactive zero-knowledge proof of knowledge (NIZKPoK).*

2.4 Predicate Encryption

The notion of *predicate-only predicate encryption* has first been introduced by Katz et al. [KSW08]. We recap the secret-key variant here. At a high level, such a scheme allows one, given the master secret key, to generate secret keys sk_f for functions f in the supported function family $\mathcal{F}$. Using the master secret key, one can further "encrypt" an attribute x to obtain a ciphertext (the "predicate-only" part in the name refers to not having a message encrypted). Decrypting the ciphertext with sk_f should then yield a constant, say 1, if $f(x) = 1$, and another constant, say 0, if $f(x) \neq 0$. Put simply, for boolean functions with range $\{0, 1\}$, this means the output of decrypt should be $f(x)$ for predicate-only PE. For simplicity, we assume that all the predicate encryption schemes that are used in this work are perfectly correct. Nevertheless, our scheme can also be instantiated using predicate encryption schemes that are not perfectly correct but this results in an additional negligible error. A formal definition of the syntax and correctness follows.

Definition 2.8 (Predicate-Only Predicate Encryption). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of sets $\mathcal{F}_\lambda$ of predicates $f \colon \mathcal{X}_\lambda \to \{0, 1\}$. A* predicate-only predicate encryption *(PE) scheme for the functionality class $\mathcal{F}_\lambda$ is a tuple of four algorithms $\mathsf{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a unary representation of the security parameter λ and outputs the master secret key msk.*

$\mathsf{KeyGen}(\mathsf{msk}, f)$**:** *Takes as input the master secret key msk and a function $f \in \mathcal{F}$, and outputs a functional key sk_f.*

$\mathsf{Enc}(\mathsf{msk}, x)$**:** *Takes as input the master secret key msk and an attribute $x \in \mathcal{X}_\lambda$, and outputs a ciphertext ct.*

$\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$**:** *Takes as input a functional key sk_f and a ciphertext ct and outputs 0 or 1.*

A predicate-only predicate encryption scheme PE is correct *if for all $\lambda \in \mathbb{N}$, for all msk in the support of $\mathsf{Setup}(1^\lambda)$, all functions $f \in \mathcal{F}_\lambda$, all secret keys sk_f in the support of $\mathsf{KeyGen}(\mathsf{msk}, f)$, and for all attributes $x \in \mathcal{X}_\lambda$, we have*

$$\Pr\big[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Enc}(\mathsf{msk}, x)) = f(x)\big] = 1.$$

As a security requirement, we want the ciphertexts to hide the encrypted attributes. We define two different security notions, indistinguishability based and simulation based, below.

Security Notions for PE. In the initial work of Katz et al. [KSW08], the authors only introduce the weaker notion of selective security, as well as a construction that achieves this notion. The corresponding indistinguishability based adaptive security notion for predicate encryption has been introduced in [OT12a]. While the definition in [OT12a] allows the adversary to obtain only a single challenge ciphertext, we use a generalization of this notion that allows for multiple challenges. A simple hybrid argument shows that our notion is implied by the secret-key variant of the definition from [OT12a]. In Appendix C, we present this single-challenge definition and provide the proof that it implies the multi-challenge definition used here.

Definition 2.9 (Indistinguishability-Based Attribute Hiding). *Let $\mathsf{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a PE scheme for a function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ as defined above. For $\beta \in \{0, 1\}$, we define the experiment $\mathrm{AH}_\beta^{\mathsf{PE}}$ in Fig. 4, where the left-or-right oracle is defined as:*

$\mathsf{QEncLR}_\beta(\cdot, \cdot)$**:** *On input two attribute sets x_0 and x_1, output $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, x_\beta)$.*

The advantage of an adversary $\mathcal{A}$ is defined as:

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

```
$\mathbf{AH}_\beta^{\mathsf{PE}}(1^\lambda, \mathcal{A})$
$\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEncLR}_\beta(\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot)}(1^\lambda)$
Output: $\alpha$
```

Fig. 4: Attribute-Hiding game of PE.

We call an adversary valid *if for all queries* (x_0, x_1) *to the oracle* $\mathsf{QEncLR}_\beta(\cdot,\cdot)$ *and for any function* f *queried to the key generation oracle* $\mathsf{KeyGen}(\mathsf{msk},\cdot)$*, we have* $f(x_0) = f(x_1)$ *(with probability* 1 *over the randomness of the adversary and the involved algorithms).*

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

The notion of semi-adaptive simulation based security has first been proposed by Wee [Wee17] and extended to several challenges [DOT18]. We formalize here a stronger definition with full adaptivity. We state this stronger definition for completeness and clarity: looking ahead, if one wants to obtain a simulation-based or UC-secure (see Section 5) PCS scheme (for some policy class), a simulation-based (predicate-only) PE scheme in this stronger sense is required (for a related set of predicates).

Definition 2.10 (Simulation-Based AH). *Let* $\mathsf{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc})$ *be a PE scheme for a function family* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$*. We define in Fig. 5 the experiments* $\mathrm{Real}^{\mathsf{PE}}(1^\lambda, \mathcal{A})$ *and* $\mathrm{Ideal}^{\mathsf{PE}}(1^\lambda, \mathcal{A}, \mathcal{S})$ *for a PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *and a PPT simulator* $\mathcal{S} = (\mathcal{S}_{\mathrm{Setup}}, \mathcal{S}_{\mathrm{Enc}}, \mathcal{S}_{\mathrm{KG}})$*. The simulator algorithms have a shared state* s*, which is modeled as giving them* s *as input, and allowing all of them to update the state* s*. Furthermore, all but* $\mathcal{S}_{\mathrm{Setup}}$ *get as an additional input the leakage set* $\mathcal{L}$*. Furthermore, the ideal encryption oracle is formally implemented by the function* $\mathcal{S}'_{\mathrm{Enc}}(s, \mathcal{L}, x) := \mathcal{S}_{\mathrm{Enc}}(s, \mathcal{L})$*. The set* $\mathcal{L}$ *is initially empty, and gets updated according to the following inductive rules:*

- *Let* n_{Enc} *be the current number of encryption queries with the corresponding values* x_i *with* $i \in [n_{\mathsf{Enc}}]$ *and* n_{KeyGen} *be the current number of key generation queries with the corresponding values* f_j *with* $j \in [n_{\mathsf{KeyGen}}]$*. Then the leakage set* $\mathcal{L}$ *contains the values of the following mapping:*
$$(i, j) \mapsto f_j(x_i)$$
for all $i \in [n_{\mathsf{Enc}}]$ *and* $j \in [n_{\mathsf{KeyGen}}]$*.*
- *On every additional encryption query* $x_{i'}$ *(where* i' *is the counter variable for encryption queries) the values*
$$(i', j) \mapsto f_j(x_{i'})$$
for all $j \in [n_{\mathsf{KeyGen}}]$ *are added to* $\mathcal{L}$*.*
- *On every additional key generation query* $f_{j'}$ *(where* j' *is the counter variable for key generation queries) the values*
$$(i, j') \mapsto f_{j'}(x_i)$$
for all $i \in [n_{\mathsf{Enc}}]$ *are added to* $\mathcal{L}$*.*

The advantage of an adversary $\mathcal{A}$ *in distinguishing both worlds is defined as*

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

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

$\mathbf{Real}^{\mathsf{PE}}(1^\lambda, \mathcal{A})$	$\mathbf{Ideal}^{\mathsf{PE}}(1^\lambda, \mathcal{A}, \mathcal{S})$
$\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$	$s \leftarrow \mathcal{S}_{\mathrm{Setup}}(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{msk},\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot)}(1^\lambda)$	$\alpha \leftarrow \mathcal{A}^{\mathcal{S}'_{\mathrm{Enc}}(s,\mathcal{L},\cdot),\mathcal{S}_{\mathrm{KG}}(s,\mathcal{L},\cdot)}(1^\lambda)$
Output: α	**Output:** α

Fig. 5: Simulation-Based Attribute Hiding of PE.

Many PE schemes are defined in the public-key setting which is not the main setting we are interested in in this work. However, any public-key PE scheme can be easily turned into a secret-key scheme by keeping both parts of the key-pair private, and hence can be of use in this work.

3 Policy-Compliant Signatures

In this section, we introduce the notion of policy-compliant signature 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 $\mathsf{QKeyGenLR}_\beta$) 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, first 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 $\mathsf{QKeyGenLR}_\beta(\cdot,\cdot)$: 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 $(i, \mathsf{pk}_i, \mathsf{sk}_i, \ldots) \in \mathcal{QK}$ for some $\mathsf{pk}_i, \mathsf{sk}_i$, then copy that entry from $\mathcal{QK}$ to $\mathcal{QC}$ and return sk_i. Otherwise, return $\perp$. [2]

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 $(i, \mathsf{pk}_i, \mathsf{sk}_i, \ldots) \in \mathcal{QK}$ 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.[3] Note that, as a further special case, 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. 6 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. 6 and define the advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *by*

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

[2] To improve readability, we use a gray color for values that appear in the lists, but are not directly relevant for the statement.

[3] 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: $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge$
        $\Big[[\exists i, \mathsf{sk}, x\ \forall j, \sigma\ (i, \mathsf{pk}, \mathsf{sk}, x) \in \mathcal{QK} \setminus \mathcal{QC} \wedge (j, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma) \notin \mathcal{QS}]$
          $\vee\ [\forall i, j, \mathsf{sk}, \mathsf{sk}^*, x_i, x^*\ (i, \mathsf{pk}, \mathsf{sk}, x_i), (j, \mathsf{pk}^*, \mathsf{sk}^*, x^*) \in \mathcal{QK} \Rightarrow F(x_i, x^*) = 0]\Big]$
```

Fig. 6: Unforgeability Game of PCS.

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:* $\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{PCS},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda)$.

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* $\mathrm{AH}^{\mathsf{PCS}}_\beta$ *in Fig. 7, 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*

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

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* $(i, \mathsf{pk}_i, \mathsf{sk}_i, x_{i,0}, x_{i,1}) \in \mathcal{QC}$ *and for all* $(j, \mathsf{pk}_j, \mathsf{sk}_j, x_{j,0}, x_{j,1}) \in \mathcal{QK}$*, 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* $(i, \mathsf{pk}_i, \mathsf{pk}_j, m, \sigma) \in \mathcal{QS}$*, and* $(i, \mathsf{pk}_i, \mathsf{sk}_i, x_{i,0}, x_{i,1}), (j, \mathsf{pk}_j, \mathsf{sk}_j, x_{j,0}, x_{j,1}) \in \mathcal{QK}$*, 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:* $\mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PCS},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda)$.

$\mathbf{AH}_\beta^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$
$(F, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PCS.Setup}(1^\lambda, F)$
$\alpha \leftarrow \mathcal{A}_2^{\mathsf{QKeyGenLR}_\beta(\cdot,\cdot),\mathsf{QCor}(\cdot),\mathsf{QSign}(\cdot,\cdot,\cdot)}(\mathsf{st}, \mathsf{mpk})$
Output: α

Fig. 7: The 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 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 Appendix B. 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. 9 to prove the above binding and correct decryption.

The full scheme is given in Fig. 8. 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. 8 (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. 9.*

```
$\underline{\mathsf{Setup}(1^\lambda, F)}$:
$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
$\mathsf{msk_{PE}} \leftarrow \mathsf{PE.Setup}(1^\lambda)$
$(\mathsf{vk_{pub}}, \mathsf{sk_{pub}}) \leftarrow \mathsf{DS_{pub}.Setup}(1^\lambda)$
$(\mathsf{vk_{priv}}, \mathsf{sk_{priv}}) \leftarrow \mathsf{DS_{priv}.Setup}(1^\lambda)$
$\mathsf{mpk} := (F, \mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}})$
$\mathsf{msk} := (\mathsf{msk_{PE}}, \mathsf{sk_{pub}}, \mathsf{sk_{priv}})$
Return $(\mathsf{mpk}, \mathsf{msk})$

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

$\underline{\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, \mathsf{pk}_R, m)}$:
Parse $\mathsf{mpk} = (F, \mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}})$
      $\mathsf{sk} = (\mathsf{vk}_S, \mathsf{sk}_S, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$
      $\mathsf{pk}_R = (\mathsf{vk}_R, \mathsf{ct}_R, \sigma_{\mathsf{pub}})$
If $\mathsf{DS_{pub}.Verify}(\mathsf{vk_{pub}}, (\mathsf{vk}_R, \mathsf{ct}_R), \sigma_{\mathsf{pub}}) = 0$
    Return $\perp$
If $\mathsf{PE.Dec}(\mathsf{sk}_{f_{\boldsymbol{x}}}, \mathsf{ct}_R) = 0$
    Return $\perp$
$\pi \leftarrow \mathsf{Prove}(\mathsf{CRS}, (\mathsf{vk_{priv}}, \mathsf{vk}_S, \mathsf{ct}_R),$
                $(\mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}}))$
      where the NIZK relation is defined in Fig. 9
$\sigma' \leftarrow \mathsf{DS_P.Sign}(\mathsf{sk}_S, (m, \mathsf{pk}_R, \pi))$
Return $\sigma := (\pi, \sigma')$

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

Fig. 8: The Policy-Compliant Signature Scheme. It uses a NIZK proof system NIZK, a predicate encryption scheme PE, and three digital signature schemes $\mathsf{DS_{pub}}, \mathsf{DS_{priv}}$ and $\mathsf{DS_P}$.

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

Fig. 9: Relation used for the PCS scheme in Fig. 8.

4.2 Correctness

The correctness of the construction described in Fig. 8 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 $\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{pk}_y \coloneqq (\mathsf{vk}_y, \mathsf{ct}_y, \sigma^y_{\mathsf{pub}}), \mathsf{sk}_x \coloneqq (\mathsf{vk}_x, \mathsf{sk}^{\mathsf{DS}}_x, \mathsf{sk}_{f_x}, \sigma^x_{\mathsf{priv}}), m)$ 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.

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 8, is existentially unforgeable. Namely, for any PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *and* $\mathcal{B}_4$ *and a distinguisher* $\mathcal{B}'$*, such that*

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{PCS},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}_1}(\lambda) + 2q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{P}},\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_4}(\lambda),$$

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

Proof. Consider the random experiment $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ for which we define the following two events:

– Event $\mathsf{KeyColl}_{\mathcal{A}}$: The adversary terminates and it holds that there are indices $i \neq j$ such that $(i, \mathsf{pk}_i, \cdot, \cdot), (j, \mathsf{pk}_j, \cdot, \cdot) \in \mathcal{QK}$, where $(\mathsf{pk}_i = (\mathsf{vk}_i, \cdot, \cdot)$, $(\mathsf{pk}_j = (\mathsf{vk}_j, \cdot, \cdot)$, for which $\mathsf{vk}_i = \mathsf{vk}_j$.
– Event $\mathsf{KeyForge}_{\mathcal{A}}$: The adversary $\mathcal{A}$ terminates with output $(\mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ and there exists an entry $(\cdot, \mathsf{pk}^*_S, \mathsf{pk}^*_R, m^*, \sigma^*) \in \mathcal{QS} \cup \{(\mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)\}$ for which the following condition holds: $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}^*_S, \mathsf{pk}^*_R, m^*, \sigma^*) = 1 \wedge ((\cdot, \mathsf{pk}^*_S, \cdot, \cdot) \notin \mathcal{QK} \vee (\cdot, \mathsf{pk}^*_R, \cdot, \cdot) \notin \mathcal{QK})$.

We denote the winning condition of the experiment by the event $\mathsf{WIN}_{\mathcal{A}}$ and split it into two parts:

– Event $\mathsf{WIN1}_{\mathcal{A}}$: The adversary generates the output $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge \exists i, \mathsf{sk}, x\ \forall j, \sigma\ (i, \mathsf{pk}, \mathsf{sk}, x) \in \mathcal{QK} \setminus \mathcal{QC} \wedge (j, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma) \notin \mathcal{QS}$.
– Event $\mathsf{WIN2}_{\mathcal{A}}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge \forall i, j, \mathsf{sk}, \mathsf{sk}^*, x_i, x^*\ (i, \mathsf{pk}, \mathsf{sk}, x_i), (j, \mathsf{pk}^*, \mathsf{sk}^*, x^*) \in \mathcal{QK} \Rightarrow F(x_i, x^*) = 0$.

By Lemma 4.4 and Lemma 4.3, we obtain

$$\Pr[\mathsf{KeyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}_1}(\lambda) \quad \text{and} \quad \Pr[\mathsf{KeyColl}_{\mathcal{A}}] \leq q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{P}},\mathcal{B}_2'}(\lambda)$$

for adversaries $\mathcal{B}_1$ and $\mathcal{B}_2'$ which are constructed based on $\mathcal{A}$ and have roughly the same efficiency as $\mathcal{A}$.

Finally, we obtain by Lemma 4.5 and by Lemma 4.6 that

$$\begin{aligned}&\Pr[\mathsf{WIN1}_{\mathcal{A}}] \leq q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{P}},\mathcal{B}_2''}(\lambda) \quad \text{and}\\ &\Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK},\mathcal{B}'},\end{aligned}$$

where the adversaries $\mathcal{B}_2''$, $\mathcal{B}_3$, $\mathcal{B}_4$, and distinguisher $\mathcal{B}'$ are constructed based on $\mathcal{A}$ and have roughly the same efficiency as $\mathcal{A}$.

By definition of the events, we have

$$\begin{aligned}\Pr[\mathsf{WIN}_{\mathcal{A}}] &\leq \Pr[\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}] + \Pr[\mathsf{WIN}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}]\\ &\leq \Pr[\mathsf{KeyColl}_{\mathcal{A}}] + \Pr[\mathsf{KeyForge}_{\mathcal{A}}] + \Pr[\mathsf{WIN1}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}]\\ &\qquad + \Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}].\end{aligned}$$

Finally, adversaries $\mathcal{B}_2'$ and $\mathcal{B}_2''$ can be combined into a single adversary $\mathcal{B}_2$ which picks $\mathcal{B} \in \{\mathcal{B}_2', \mathcal{B}_2''\}$ at random and running it against EUF-CMA$^{\mathsf{DS}_{\mathsf{P}}}$. The theorem follows. □

Lemma 4.3. *Consider the experiment* **EUF-CMA**$^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ *and Let* $\mathsf{KeyForge}_{\mathcal{A}}$ *be defined as above. We construct an adversary* $\mathcal{B}$ *such that* $\Pr[\mathsf{KeyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}}(\lambda)$.

Proof. We build an adversary $\mathcal{B}$ that simulates EUF-CMA$^{\mathsf{PCS}}$ towards $\mathcal{A}$ when interacting with the underlying strong unforgeability experiment SUF-CMA$^{\mathsf{DS}_{\mathsf{pub}}}$ and show that if $\mathcal{A}$ outputs $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ as described in the above event, it can be used as a forgery in the SUF-CMA$^{\mathsf{DS}_{\mathsf{pub}}}$ experiment.

In the first step, the adversary $\mathcal{B}$ receives $\mathsf{vk}_{\mathsf{pub}}$ from the forgeability experiment SUF-CMA$^{\mathsf{DS}_{\mathsf{pub}}}$ and the policy F from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}$ generates $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}(1^\lambda)$, $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and $\mathsf{msk}_{\mathsf{PE}} \leftarrow \mathsf{PE.Setup}(1^\lambda)$, sets $\mathsf{mpk} := (F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}})$ and sends mpk to the adversary $\mathcal{A}$. The adversary $\mathcal{B}$ also initializes the lists $\mathcal{QK} = \{\}$ and $\mathcal{QC} = \{\}$ and the counter $i = 1$.

Whenever $\mathcal{A}$ asks a query x to the key-generation oracle $\mathsf{QKeyGen}$, the adversary $\mathcal{B}$ samples a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$, generates the subpolicy and attribute vector $(f_{\boldsymbol{x}}, \boldsymbol{x}) = \mathsf{SubPol}(F, x)$ of F using the attribute set x, generates an encryption of the attribute vector $\boldsymbol{x}$, $\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mathsf{msk}_{\mathsf{PE}}, \boldsymbol{x})$ and generates a secret key for the subpolicy $f_{\boldsymbol{x}}$, i.e. $\mathsf{sk}_{f_{\boldsymbol{x}}} \leftarrow \mathsf{PE.KeyGen}(\mathsf{msk}_{\mathsf{PE}}, f_{\boldsymbol{x}})$, and generates the signature $\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}))$. In the next step, the adversary $\mathcal{B}$ submits $(\mathsf{vk}_{\mathsf{P}}, \mathsf{ct})$ to its signing oracle and receives σ_{pub} as a reply. Then, $\mathcal{B}$ sets $\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$, adds $(i, \mathsf{pk}, \mathsf{sk}, x)$ to $\mathcal{QK}$, increases the counter i, i.e. $i := i + 1$, and sends pk to $\mathcal{A}$.

For every query j to the corruption oracle QCor, $\mathcal{B}$ returns sk to $\mathcal{A}$ for $(j, \mathsf{pk}, \mathsf{sk}, x) \in \mathcal{QK}$, and returns $\perp$ if no such entry exists. It then adds $(j, \mathsf{pk}, \mathsf{sk}, x)$ to the list $\mathcal{QC}$.

If the adversary $\mathcal{A}$ asks a query $(j, \mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, \sigma^R_{\mathsf{pub}}), m)$ to the signing oracle QSign, the adversary $\mathcal{B}$ executes $\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, \mathsf{pk}_R, , m)$ with $(j, \cdot, \mathsf{sk}, \cdot) \in \mathcal{QK}$ and sends the output as a reply to $\mathcal{A}$. If no entry $(j, \cdot, \cdot, \cdot) \in \mathcal{QK}$ exists, it outputs $\perp$.

We observe that the simulation of $\mathcal{B}$ towards $\mathcal{A}$ is perfect since the only difference is the generation of the verification key $\mathsf{vk}_{\mathsf{pub}}$ and the corresponding signatures, which, in this setting, are honestly generated by the underlying challenger, which results in the identical distribution as in the PCS experiment.

Thus, when $\mathcal{A}$ terminates with output $(\mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$, adversary $\mathcal{B}$ considers the set of all tuples $\mathcal{QS} \cup \{(\mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)\}$ and searches for a tuple $(\mathsf{pk}^*_S := (\mathsf{vk}^*_S, \mathsf{ct}^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_R, \mathsf{ct}^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^*)$ that fulfills the condition of the event $\mathsf{KeyForge}_{\mathcal{A}}$, that is, the condition $(\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}^*_S, \mathsf{pk}^*_R, m^*, \sigma^*) = 1 \wedge ((\cdot, \mathsf{pk}^*_S, \cdot, \cdot) \notin \mathcal{QK} \vee (\cdot, \mathsf{pk}^*_R, \cdot, \cdot)) \notin \mathcal{QK}) = 1$. If no such tuple exists, the adversary aborts with output $\perp$. Otherwise, at least one of the two message-signature pairs $((\mathsf{vk}^*_S, \mathsf{ct}^*_S), \sigma^*_{\mathsf{pub},S})$ or $((\mathsf{vk}^*_R, \mathsf{ct}^*_R), \sigma^*_{\mathsf{pub},R})$ has to be a forgery the underlying SUF-CMA$^{\mathsf{DS_{pub}}}$ experiment w.r.t. public key $\mathsf{vk_{pub}}$. Without loss of generality, let $(\cdot, \mathsf{pk}^*_S, \cdot, \cdot) \notin \mathcal{QK}$ be the tuple fulfilling the above condition. This is a valid forgery, since either the pair $(\mathsf{vk}^*_S, \mathsf{ct}^*_S)$ has never been queried to the signing oracle of SUF-CMA$^{\mathsf{DS_{pub}}}$ by $\mathcal{B}$, or, if it has been queried, the result was not $\sigma^*_{\mathsf{pub},S}$ as otherwise we would have $(\cdot, \mathsf{pk}^*_S, \cdot, \cdot) \in \mathcal{QK}$. This concludes the proof. □

Lemma 4.4. *Consider the experiment* $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ *and Let* $\mathsf{KeyColl}_{\mathcal{A}}$ *be defined as above. We construct an adversary* $\mathcal{B}$ *such that* $\Pr[\mathsf{KeyColl}_{\mathcal{A}}] \leq q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS_P},\mathcal{B}}(\lambda)$.

Proof. We observe that in the the execution of $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ all public keys added to the set $\mathcal{QK}$ are computed by calling $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \cdot)$, where $(\mathsf{pk}_i = (\mathsf{vk}_i, \cdot, \cdot)$ and $(\mathsf{vk}_i, \cdot) \leftarrow \mathsf{DS_P.Setup}(1^\lambda)$. At the point when $\mathcal{A}$ terminates, by definition of the event, we have two indices i, j such that in particular $\mathsf{vk}_i = \mathsf{vk}_j$ and $i \neq j$. We further see that $\Pr[\mathsf{KeyColl}_{\mathcal{A}}] \leq \sum_{k=1}^{q} \Pr[\exists j : \mathsf{vk}_k = \mathsf{vk}_j]$, which follows from the union bound. Since the distribution of keys is independent of the index, denote $\alpha := \Pr[\exists j : \mathsf{vk}_k = \mathsf{vk}_j]$ for some arbitrarily fixed index k. We now construct an adversary $\mathcal{B}$ against the signature scheme $\mathsf{DS_P}$: on input a verification key $\tilde{\mathsf{vk}}$ from its challenger, $\mathcal{B}$ samples $q-1$ PCS public keys. This induces the same distribution as the distribution of the vk_i's in $\mathcal{QK}$. Therefore, $\Pr[\exists j : \tilde{\mathsf{vk}} = \mathsf{vk}_j] \geq \alpha$. Conditioned on this event, $\mathcal{B}$ can forge a signature with probability 1 since it knows the secret key corresponding to vk_j and because of the correctness of the signature scheme, we can generate fresh signature for any message m that will successfully verify w.r.t. $\tilde{\mathsf{vk}}$. □

Lemma 4.5. *Consider the experiment* $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ *and Let* $\mathsf{WIN1}_{\mathcal{A}}$ *be defined as above. We construct an adversary* $\mathcal{B}$ *such that* $\Pr[\mathsf{WIN1}_{\mathcal{A}}] \leq q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS_P},\mathcal{B}}(\lambda)$.

Proof. We define the following events $\mathsf{E}_1, \ldots, \mathsf{E}_q$: The event E_k is the event where the adversary outputs a valid signature for the k'th public key output by the generation oracle $\mathcal{QK}$ not obtained from the signing oracle without querying it to the corruption oracle QCor. Formally, the adversary $\mathcal{A}$ outputs $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ in the EUF-CMA$^{\mathsf{PCS}}$ game such that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$, $(\mathsf{pk}, \mathsf{pk}^*, m^*, \cdot) \notin \mathcal{QS}$, $(k, \mathsf{pk}, \cdot, \cdot) \notin \mathcal{QC}$, and $(k, \mathsf{pk}, \cdot, \cdot) \in \mathcal{QK}$.

When q denotes the number of queries to $\mathsf{QKeyGen}$, we see that $\mathsf{WIN1}_{\mathcal{A}} = \bigcup_{k=1}^{q} E_k$, since by definition, the public key pk^*_S specified in a forgery output by an adversary $\mathcal{A}$ corresponds to some index of the set $\mathcal{QK} \cup \mathcal{QC}$ and at least one index must not be corrupted.

We now bound $\Pr[E_k]$. We build an adversary $\mathcal{B}_k$ that simulates EUF-CMA$^{\mathsf{PCS}}$ towards $\mathcal{A}$ when interacting with the underlying existential unforgeability experiment EUF-CMA$^{\mathsf{DS_P}}$ and show that if $\mathcal{A}$ outputs $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ as described in event E_k it can be used as a forgery in the EUF-CMA$^{\mathsf{DS_P}}$ experiment.

In the first step, the adversary $\mathcal{B}_k$ receives $\mathsf{vk}^{\mathsf{chall}}_{\mathsf{P}}$ from experiment EUF-CMA$^{\mathsf{DS_P}}$ and the policy F from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}_k$ generates $(\mathsf{vk_{pub}}, \mathsf{sk_{pub}}) \leftarrow \mathsf{DS_{pub}.Setup}(1^\lambda)$, $(\mathsf{vk_{priv}}, \mathsf{sk_{priv}}) \leftarrow \mathsf{DS_{priv}.Setup}(1^\lambda)$, $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and $\mathsf{msk_{PE}} \leftarrow \mathsf{PE.Setup}(1^\lambda)$, sets $\mathsf{mpk} := (F, \mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}})$ and sends mpk to the adversary $\mathcal{A}$. The adversary $\mathcal{B}_k$ also initializes the lists $\mathcal{QK} = \{\}$ and $\mathcal{QC} = \{\}$ and the counter $i = 1$.

For a query x to the key generation oracle $\mathsf{QKeyGen}$, we distinguish between two cases. The query is not the k'th query to the oracle, i.e. $i \neq k$, and the query is the k'th query to the oracle, i.e. $i = k$. In the first case, $\mathcal{B}_k$ samples a signature key pair $(\mathsf{vk}_\mathsf{P}, \mathsf{sk}_\mathsf{P}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$, generates the subpolicy and attribute vector $(f_{\boldsymbol{x}}, \boldsymbol{x}) = \mathsf{SubPol}(F, x)$ of F using the attribute set x, generates an encryption of the attribute vector $\boldsymbol{x}$, $\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mathsf{msk}_\mathsf{PE}, \boldsymbol{x})$ and generates a secret key for the subpolicy $f_{\boldsymbol{x}}$, $\mathsf{sk}_{f_{\boldsymbol{x}}} \leftarrow \mathsf{PE.KeyGen}(\mathsf{msk}_\mathsf{PE}, f_{\boldsymbol{x}})$. In the next step, the adversary $\mathcal{B}_k$ generates the signatures σ_pub and signature σ_priv, where $\sigma_\mathsf{pub} \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{pub}, (\mathsf{vk}_\mathsf{P}, \mathsf{ct}))$ and $\sigma_\mathsf{priv} \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{priv}, (\mathsf{vk}_\mathsf{P}, \mathsf{sk}_{f_{\boldsymbol{x}}}))$. Then, $\mathcal{B}_k$ sets $\mathsf{pk} := (\mathsf{vk}_\mathsf{P}, \mathsf{ct}, \sigma_\mathsf{pub})$ and $\mathsf{sk} := (\mathsf{vk}_\mathsf{P}, \mathsf{sk}_\mathsf{P}, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_\mathsf{priv})$, adds $(i, \mathsf{pk}, \mathsf{sk}, x)$ to $\mathcal{QK}$, increases the counter i, i.e. $i := i + 1$, and sends pk to $\mathcal{A}$. If the query is the k'th query to the key generation oracle $\mathsf{QKeyGen}$, $\mathcal{B}$ proceeds in the same way as in the first case, with the only difference that instead of generating a fresh signature key pair, it uses the verification key $\mathsf{vk}_\mathsf{P}^\mathsf{chall}$. Afterwards, $\mathcal{B}_k$ sets $\mathsf{pk} := (\mathsf{vk}_\mathsf{P}^\mathsf{chall}, \mathsf{ct}, \sigma_\mathsf{pub})$ and $\mathsf{sk} := (\mathsf{vk}_\mathsf{P}^\mathsf{chall}, \cdot, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_\mathsf{priv})$, adds $(k, \mathsf{pk}, \mathsf{sk}, x)$ to $\mathcal{QK}$, increases the counter i, i.e. $i := i + 1$, and sends pk to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks a corruption query j to the corruption oracle QCor, the adversary $\mathcal{B}_k$ checks the list $\mathcal{QK}$ to find $(j, \mathsf{pk}, \mathsf{sk}, x)$. If no such entry exists, the adversary $\mathcal{B}_k$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk to $\mathcal{A}$ and adds $(j, \mathsf{pk}, \mathsf{sk}, x)$ to $\mathcal{QC}$. In the case that $\mathcal{A}$ asks a corruption query for the k'th key, the adversary $\mathcal{B}_k$ aborts the execution (note that in this case, E_k does not occur).

If the adversary $\mathcal{A}$ asks a query $(j, \mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, \sigma_{\mathsf{pub},R}), m)$ to the sign oracle QSign, the adversary $\mathcal{B}_k$ checks that $(j, \cdot, \cdot, \cdot) \in \mathcal{QK}$ and returns $\perp$ if this is not the case. Now, we distinguish between two cases, first the signature is requested for the k'th public key, i.e. $j = k$, and, second, the signature is not requested the k'th public key, i.e. $j \neq k$. In the first case, $\mathcal{B}_k$ checks that $\mathsf{PE.Dec}(\mathsf{sk}_{f_{\boldsymbol{x}}}, \mathsf{ct}_R) = 1$, computes $\pi \leftarrow \mathsf{Prove}(\mathsf{CRS}, (\mathsf{vk}_\mathsf{priv}, \mathsf{vk}_S, \mathsf{ct}_R), (\mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_\mathsf{priv}))$ with $(k, \cdot, \mathsf{sk} := (\mathsf{vk}_S, \cdot, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_\mathsf{priv}), x) \in \mathcal{QK}$, submits (m, pk_R, π) to the signature oracle of EUF-CMA$^{\mathsf{DS}_\mathsf{P}}$ and receives σ' as a reply. Then, $\mathcal{B}_k$ sets $\sigma := (\pi, \sigma')$ and outputs σ to $\mathcal{A}$. In the second case, $\mathcal{B}_k$ executes Sign using sk with $(j, \cdot, \mathsf{sk}, \cdot) \in \mathcal{QK}$ and sends the resulting signature σ as a reply to $\mathcal{A}$.

Finally, when $\mathcal{A}$ terminates with output $(\mathsf{pk}_S^* := (\mathsf{vk}_S^*, \mathsf{ct}_S^*, \sigma_{\mathsf{pub},S}^*), \mathsf{pk}_R^* := (\mathsf{vk}_R^*, \mathsf{ct}_R^*, \sigma_{\mathsf{pub},R}^*), m^*, \sigma^* := (\pi^*, \sigma'))$ check the conditions of E_k, i.e., $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S^*, \mathsf{pk}_R^*, m^*, \sigma^*) = 1$ with $(\mathsf{pk}_S^*, \mathsf{pk}_R^*, m^*, \cdot) \notin \mathcal{QS}$, $(k, \mathsf{pk}_S^*, \cdot, \cdot) \notin \mathcal{QC}$ and verify that indeed $(k, \mathsf{pk}_S^*, \cdot, \cdot) \in \mathcal{QK}$, in which case $\mathsf{pk}_S^* = (\mathsf{vk}_\mathsf{P}^\mathsf{chall}, \dots)$. In this case the adversary $\mathcal{B}_k$ outputs $((m^*, \mathsf{pk}_R^*, \pi^*), \sigma')$ as its forgery to the underlying EUF-CMA$^{\mathsf{DS}_\mathsf{P}}$ experiment.

We first observe that if $\mathcal{B}_k$ does not abort and the conditions of event E_k hold in this emulation then $\mathcal{B}_k$'s output is a valid forgery and thus wins in experiment EUF-CMA$^{\mathsf{DS}_\mathsf{P}}$: by definition of E_k, we must in particular have a valid signature for $(m^*, \mathsf{pk}_R^*, \pi^*)$ w.r.t. $\mathsf{vk}_\mathsf{P}^\mathsf{chall}$ specified in pk_S^*. If m^* was never part of any query by $\mathcal{A}$ to the emulated singing oracle QSign for signer index k, then the output of $\mathcal{B}_k$ in experiment EUF-CMA$^{\mathsf{DS}_\mathsf{P}}$ is a novel message. If m^* has been asked, then it holds that $\mathcal{B}_k$ did not emulate a signing operation queried by $\mathcal{A}$ for m^* specifically for the receiver public key pk_R^* as the combination would contradict event E_k. Hence, the combination (m, pk_R^*) must be novel.

Second, we observe that the simulation of $\mathcal{B}_k$ towards $\mathcal{A}$ is perfect up to the point where it aborts. The distribution of keys that $\mathcal{A}$ observes are $|\mathcal{QK}|$ independently and honestly sampled public keys (and correctly computed signatures thereof) and the emulated oracle calls execute the scheme as in the experiment $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$, thus the definition of event E_k applies directly to the emulation and the probability is the same as in an execution of $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$.

Therefore, the probability that $\mathcal{B}_k$ terminates with a valid forgery is $\mathsf{Adv}_{\mathsf{DS}_\mathsf{P}, \mathcal{B}_k}^{\text{EUF-CMA}}(\lambda) = \Pr[E_k]$. This concludes the analysis for $\mathcal{B}_k$. We obtain the lemma statement by picking an index $k = 1 \dots q$ at random and executing $\mathcal{B}_k$. $\square$

Lemma 4.6. *Consider the experiment* $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ *and Let* $\mathsf{WIN2}_{\mathcal{A}}$ *be defined as above. We can construct adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *and a distinguisher* $\mathcal{B}'$ *such that* $\Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}] \leq \mathsf{Adv}^{\mathrm{CRS}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\mathrm{EUF\text{-}CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda)$.

Proof. On a high-level, the adversary needs to prove a wrong claim which can either be done by attacking the NIZK directly, or if the NIZK is extractable, then the attacker must attack the underlying signature scheme in order to possess a valid witness.

We first make a first transition to a hybrid world $\text{EUF-CMA}^{\mathsf{PCS}}_{\text{Hyb}}$, which is identical to $\text{EUF-CMA}^{\mathsf{PCS}}$ except that we replace $\mathsf{NIZK.Setup}(1^\lambda)$ by the CRS simulation algorithm E_1 associated to the NIZK scheme. All above defined events are still defined in this hybrid experiment Clearly, we can construct a distinguisher $\mathcal{B}'$ such that

$$\begin{aligned}\Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}] \\ \leq \Pr_{\text{Hyb}}\left[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}\right] + \mathsf{Adv}^{\mathrm{CRS}}_{\mathsf{NIZK},\mathcal{B}'},\end{aligned}$$

where $\Pr_{\text{Hyb}}[.]$ makes explicit that this probability is taken w.r.t. experiment $\text{EUF-CMA}^{\mathsf{PCS}}_{\text{Hyb}}$. This reduction is standard: in order to distinguish the two distributions, on input a sample CRS, the distinguisher $\mathcal{B}'$ emulates the experiment towards $\mathcal{A}$. When $\mathcal{A}$ terminates, $\mathcal{B}'$ outputs 1 if event $\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}$ occurs (which is computable by $\mathcal{B}'$ that manages all key-sets).

We build an adversary $\mathcal{B}_1$ that simulates $\text{EUF-CMA}^{\mathsf{PCS}}_{\text{Hyb}}$ towards $\mathcal{A}$ when interacting with the underlying $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{priv}}}$ experiment. We show that if $\mathcal{A}$ outputs $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ as event $\mathsf{WIN2}$ defines it can be used as a forgeability attack in the $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{priv}}}$ experiment unless a certain failure event $\mathsf{Fail}_{\mathsf{ext}}$ occurs in the reduction, which we then relate to the extraction advantage.

In the first step, the adversary $\mathcal{B}_1$ receives $\mathsf{vk}_{\mathsf{priv}}$ from the $\text{EUF-CMA}^{\mathsf{priv}}$ experiment and the policy F from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}_1$ generates $\mathsf{CRS} \leftarrow E_1(1^\lambda)$, $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}(1^\lambda)$ and $\mathsf{msk}_{\mathsf{PE}} \leftarrow \mathsf{PE.Setup}(1^\lambda)$, sets $\mathsf{mpk} := (F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}})$ and sends mpk to the adversary $\mathcal{A}$. The adversary $\mathcal{B}_1$ also initializes the lists $\mathcal{QK} = \{\}$ and $\mathcal{QC} = \{\}$ and the counter $i = 1$.

For a query x to the key generation oracle $\mathsf{QKeyGen}$, $\mathcal{B}_1$ samples a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$, generates the subpolicy and attribute vector $(f_{\boldsymbol{x}}, \boldsymbol{x}) = \mathsf{SubPol}(F, x)$ of F using the attribute set x, generates an encryption of the attribute vector $\boldsymbol{x}$, $\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mathsf{msk}_{\mathsf{PE}}, \boldsymbol{x})$ and generates a secret key for the subpolicy $f_{\boldsymbol{x}}$, $\mathsf{sk}_{f_{\boldsymbol{x}}} \leftarrow \mathsf{PE.KeyGen}(\mathsf{msk}_{\mathsf{PE}}, f_{\boldsymbol{x}})$. In the next step, the adversary $\mathcal{B}_1$ generates the signatures $\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}))$ and submits $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}})$ to the sign oracle of the underlying $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{priv}}}$ experiment to obtain the signature σ_{priv}. Then, $\mathcal{B}_1$ sets $\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$, adds $(i, \mathsf{pk}, \mathsf{sk}, x)$ to $\mathcal{QK}$, increases the counter i, i.e. $i := i + 1$, and sends pk to $\mathcal{A}$. If at any point in time, the conditions of event $\mathsf{KeyColl}_{\mathcal{A}}$ are fulfilled, $\mathcal{B}_1$ aborts.

Whenever $\mathcal{A}$ asks a corruption query j to the corruption oracle QCor, adversary $\mathcal{B}_1$ checks the list $\mathcal{QK}$ to find $(j, \mathsf{pk}, \mathsf{sk}, x)$. If no such entry exists, the adversary $\mathcal{B}_1$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk to $\mathcal{A}$ and adds $(j, \mathsf{pk}, \mathsf{sk}, x)$ to $\mathcal{QC}$.

If the adversary $\mathcal{A}$ asks a query $(j, \mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, \sigma^R_{\mathsf{pub}}), m)$ to the sign oracle QSign, the adversary $\mathcal{B}_1$ checks that $(j, \cdot, \cdot, \cdot) \in \mathcal{QK}$ and that pk_R has been output by the key generation oracle. If this is not the case the adversary $\mathcal{B}_1$ aborts. Otherwise, it continues with the execution of Sign, using the key sk where $(j, \cdot, \mathsf{sk}, \cdot) \in \mathcal{QK}$. Finally, $\mathcal{B}_1$ sends the resulting signature σ, containing π, as a reply to $\mathcal{A}$.

When $\mathcal{A}$ terminates with $(\mathsf{pk}^*_S := (\mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_{\mathsf{P},R}, \mathsf{ct}^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^*, \sigma'))$ $\mathcal{B}_1$ first checks whether the conditions of event $\mathsf{KeyForge}_{\mathcal{A}}$ hold, in which case it aborts.

It also verifies that the conditions of event $\mathsf{WIN2}_{\mathcal{A}}$ holds and in case this is true, it first calls $(\mathsf{sk}^*_{\mathrm{pe},S}, \sigma^*_{\mathsf{priv},S}) \leftarrow E_2(\mathsf{CRS}, (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_R), \pi^*)$ and checks whether $(x := (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_R), w := (\mathsf{sk}^*_{\mathrm{pe},S}, \sigma^*_{\mathsf{priv},S})) \in R_{\mathrm{ZK}}$ (which is efficiently checkable) and if this is the case, it submits $((\mathsf{vk}^*_{\mathsf{P},S}, \mathsf{sk}^*_{\mathrm{pe},S}), \sigma^*_{\mathsf{priv},S})$ as a forgery to the underlying experiment $\mathrm{EUF\text{-}CMA}^{\mathsf{DS}_{\mathsf{priv}}}$. If $(x, w) \notin R_{\mathrm{ZK}}$ then abort with failure event $\mathsf{Fail}_{\mathrm{ext}}$.

We observe that the emulation towards adversary $\mathcal{A}$ is perfect until the point in the execution where $\mathcal{B}_1$ would abort. The only difference is the generation of the verification key $\mathsf{vk}_{\mathsf{priv}}$ and the corresponding signatures, which, in this setting, are all honestly generated by the underlying challenger. Therefore, all defined events for experiment $\mathrm{EUF\text{-}CMA}^{\mathsf{PCS}}_{\mathrm{Hyb}}$ are likewise defined in this emulation and with respectively the same probabilities.

We now analyze the final forgery output of a run of $\mathcal{B}_1$ (which therefore does not abort). In this case, we observe that all signature verification keys are unique and that all keys can be uniquely associated to some attributes as all keys including the output pk^*_S and pk^*_R of $\mathcal{A}$ are in the key set $\mathcal{QK}$. Therefore, entry $(i, \mathsf{pk}_i = (\mathsf{vk}_i, \mathsf{ct}_i, \cdot), \mathsf{sk}_i, x_i)$ associates x_i with vk_i and there are indices j, k such that $\mathsf{pk}_j = \mathsf{pk}^*_S$ and $\mathsf{pk}_k = \mathsf{pk}^*_R$. Let us fix these two indices. Furthermore, we can assume that $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_j, \mathsf{ct}_k), \pi^*) = 1$ as otherwise, $\mathsf{WIN2}$ would not hold. Additionally, we know that that $sk^*_{\mathrm{pe},S}$ is a correct witness, in particular, $\mathsf{PE.Dec}(\mathsf{sk}^*_{\mathrm{pe},S}, \mathsf{ct}_k) = 1$. However, we know that $F(x_j, x_k) = 0$ and since the predicate-encryption scheme is perfectly correct and all keys and ciphertexts are generated honestly by $\mathcal{B}$, the entry $(j, \mathsf{pk}_j, \mathsf{sk}_j = (\cdot, \cdot, \mathsf{sk}_{f_{\boldsymbol{x}_j}}, \sigma_{\mathsf{priv},j}), x_j)$ specifies $\mathsf{sk}_{f_{\boldsymbol{x}_j}}$ for which $\mathsf{PE.Dec}(\mathsf{sk}_{f_{\boldsymbol{x}_j}}, \mathsf{ct}_k) = 0$, and thus $\mathsf{sk}_{f_{\boldsymbol{x}_j}} \neq \mathsf{sk}^*_{\mathrm{pe},S}$. Since by uniqueness, $\mathcal{B}_1$ has only submitted the query $(\mathsf{vk}_j, \mathsf{sk}_{f_{\boldsymbol{x}_j}})$ to the signing oracle of $\mathrm{EUF\text{-}CMA}^{\mathsf{DS}_{\mathsf{priv}}}$, the pair $((\mathsf{vk}^*_{\mathsf{P},S} = \mathsf{vk}_j, \mathsf{sk}^*_{\mathrm{pe},S} \neq \mathsf{sk}_{f_{\boldsymbol{x}_j}}), \sigma^*_{\mathsf{priv},S})$ is a valid signature for a novel message and therefore, a valid forgery. We obtain that the probability that $\mathcal{B}_1$ terminates with a valid forgery is therefore

$$\mathsf{Adv}^{\mathrm{EUF\text{-}CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1} = \Pr_{\mathrm{Hyb}}\left[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\mathrm{ext}}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}\right].$$

We obtain

$$\Pr_{\mathrm{Hyb}}\left[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}\right] = \mathsf{Adv}^{\mathrm{EUF\text{-}CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1} + \underbrace{\Pr_{\mathrm{Hyb}}\left[\mathsf{WIN2}_{\mathcal{A}} \cap \mathsf{Fail}_{\mathrm{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}\right]}_{\leq \mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}}.$$

It is straightforward to obtain an adversary $\mathcal{B}_2$ (based on $\mathcal{A}$) which has an advantage $\mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{B}_2} = \Pr_{\mathrm{Hyb}}\left[\mathsf{WIN2}_{\mathcal{A}} \cap \mathsf{Fail}_{\mathrm{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}\right]$. In fact, the adversary $\mathcal{B}_2$ receives as input the CRS and executes the same instructions as $\mathcal{B}_1$, with the exceptions that it can simply generate signatures for the scheme $\mathsf{DS}_{\mathsf{priv}}$ by itself. In addition, when $\mathcal{A}$ terminates with output $(\mathsf{pk}^*_S := (\mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_{\mathsf{P},R}, \mathsf{ct}^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}_2$ behaves as $\mathcal{B}_1$ but does not execute the final steps running the extractor, but instead just outputs $(x := (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_R), \pi^*)$ in case the conditions of $\mathsf{WIN2}$ are satisfied (note that the extractor is run as part of the experiment in Definition 2.7). As above, the emulation toward $\mathcal{A}$ is perfect until the point $\mathcal{B}_2$ would abort. Therefore, the advantage is as claimed, because the event of interest is that the extractor E_2 is called precisely on the accepting proof string π^* output by $\mathcal{A}$ (which is accepting for statement x as defined above because of event $\mathsf{WIN2}$) but the extraction produces a witness w such that $(x, w) \notin R_{\mathrm{ZK}}$. The statement follows. □

4.4 Indistinguishability-Based Attribute Hiding

We next prove that our PCS scheme is attribute hiding.

Theorem 4.7. *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 (for the relation* R_{ZK} *of Fig. 9) and let* $\mathsf{DS_{pub}} = (\mathsf{DS_{pub}.Setup}, \mathsf{DS_{pub}.Sign}, \mathsf{DS_{pub}.Verify})$ *be a strongly unforgeable signature scheme, then the construction* $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figure 8, is attribute hiding. Namely, for any valid PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}, \mathcal{B}'$ *and* $\mathcal{B}''$*, such that:*

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

Proof. To prove this statement, we use a hybrid argument where the games are defined as follows:

Game G_0: This game is defined as $\mathrm{AH}_0^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$.

Game G_1: In this game, we change the behavior of the sign oracle QSign and define a modified sign oracle QSign'. The oracle QSign' is defined as QSign with the difference that it only answers queries for receiver keys that have previously been output by the key generation oracle $\mathsf{QKeyGenLR}_0$, i.e. for a query (i, pk', m) with $(i, \cdot, \cdot, \cdot, \cdot) \notin \mathcal{QK}$ or $(\cdot, \mathsf{pk}', \cdot, \cdot, \cdot) \notin \mathcal{QK}$ the sign oracle QSign' outputs $\perp$. The transition from G_0 to G_1 is justified by the same reasoning as we have seen in Lemma 4.3 (bounding event $\mathsf{KeyForge}_{\mathcal{A}}$). More precisely, in Lemma 4.8, we exhibit a PPT adversary $\mathcal{B}_0$ such that:

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

Game G_2: In this game, we change from an honestly generated CRS and honestly generated proofs to a simulated CRS and simulated proofs. That is, upon a PCS signing query (j, pk_R, m), we find the attributes $x_{j,0}$ and $x_{k,0}$ s.t. $F(x_{j,0}, x_{k,0}) = 1$ ($x_{k,0}$ is associated with the key-generation event that produced pk_R) and then we simulate the proof using the NIZK simulator on input the trapdoor and $(\mathsf{vk_{priv}}, \mathsf{vk}_j, \mathsf{ct}_R)$ (note that all values are defined since by definition of this hybrid, all keys for which a signature is returned, have been generated using the key-generation oracle. In any other case (in particular associated attributes do not satisfy the policy), we output $\perp$. The transition from G_1 to G_2 is justified by the zero-knowledge property of NIZK. Namely, in Lemma 4.9, we exhibit a PPT adversary $\mathcal{B}_1$ such that:

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

Game G_3: 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. Similarly, upon PCS signing, we now find the attributes $x_{j,1}$ and $x_{k,1}$ s.t. $F(x_{j,1}, x_{k,1}) = 1$ before simulating the proofs as above. The transition from G_2 to G_3 is justified by the attribute-hiding property of PE. Namely, in Lemma 4.10, we exhibit a PPT adversary $\mathcal{B}_2$ such that:

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

Game G_4: In this game, we change back from a simulated CRS and simulated proofs π to an honestly generated CRS and honestly generated proofs π. Symmetrical to Lemma 4.9, this transition is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $\mathcal{B}_3$ such that:

$$|\Pr[\mathsf{G}_3(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{G}_4(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_3}(\lambda).$$

Game G_5: This game is the $\mathrm{AH}_1^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ game. In this game, we change the behavior of the signing oracle back from QSign' to QSign. As in Lemma 4.8, this transition is justified again by the inability to forge public keys, i.e., we can exhibit a PPT adversary $\mathcal{B}_4$ such that:

$$|\Pr[\mathsf{G}_4(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{G}_5(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{SUF\text{-}CMA}}_{\mathsf{DS_{pub}},\mathcal{B}_4}(\lambda).$$

From the definition of the games it is clear that

$$\begin{aligned}\mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PCS},\mathcal{A}}(1^\lambda) &= |\Pr[\mathrm{AH}^{\mathsf{PCS}}_0(1^\lambda,\mathcal{A})=1] - \Pr[\mathrm{AH}^{\mathsf{PCS}}_1(1^\lambda,\mathcal{A})=1]| \\ &= |\Pr[\mathsf{G}_0(\lambda,\mathcal{A})=1] - \Pr[\mathsf{G}_5(\lambda,\mathcal{A})=1]|,\end{aligned}$$

and hence the theorem follows. □

Lemma 4.8 (Transition from G_0 to G_1). *For any valid PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_0$ such that*

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

Proof (Sketch). As described above, the difference between the games G_0 and G_1 is that in the game G_0 the adversary $\mathcal{A}$ has access to the sign oracle QSign (defined in Section 3.1) and in the game G_1 the adversary $\mathcal{A}$ has access to the sign oracle QSign', which we informally described above and which is formally defined as:

$\mathsf{QSign}'(i,\mathsf{pk}',m)$**:** On input a (sender) index i, a (receiver) public key pk', and a message m, if $\mathcal{QK}$ contains an entry $(i,\mathsf{pk},\mathsf{sk},x_0,x_1) \in \mathcal{QK}$ and an entry $(j,\mathsf{pk}',\mathsf{sk}',x'_0,x'_1) \in \mathcal{QK}$, then return $\sigma \leftarrow \mathsf{PCS.Sign}(\mathsf{mpk},\mathsf{sk},\mathsf{pk}',m)$ and add $(i,\mathsf{pk},\mathsf{pk}',m,\sigma)$ to $\mathcal{QS}$. Otherwise, return $\perp$.

Compared to the oracle QSign', the signing oracle QSign does not require the receiver key pk' to have been previously output by the challenger, i.e. $(\cdot,\mathsf{pk}',\cdot,\cdot,\cdot) \notin \mathcal{QK}$, to obtain as a reply a valid signature $\sigma \neq \perp$. This is not possible for the oracle QSign' where every query using a receiver key pk' that has not been generated by the challenger, i.e. $(\cdot,\mathsf{pk}',\cdot,\cdot,\cdot) \notin \mathcal{QK}$, results in an invalid signature $\sigma = \perp$.

Therefore, to show that the games G_0 and G_1 are indistinguishable, it suffices to show that the probability that the adversary queries the signing oracle QSign using a receiver key pk' that has not been previously generated by the challenger, i.e. $(\cdot,\mathsf{pk}',\cdot,\cdot,\cdot) \notin \mathcal{QK}$, and that leads to a valid signature $\sigma \neq \perp$ is negligible. We denote this as the event $\mathsf{SignForge}_\mathcal{A}$.

For the event $\mathsf{SignForge}_\mathcal{A}$ to occur, the adversary $\mathcal{A}$ needs to generate a receiver key that verifies with respect to the signature scheme $\mathsf{DS_{pub}}$, i.e., it needs to generate a key $\mathsf{pk}' := (\mathsf{vk}',\mathsf{ct}',\sigma'_{\mathsf{pub}})$ such that $\mathsf{DS_{pub}.Verify}(\mathsf{vk_{pub}},(\mathsf{vk}',\mathsf{ct}'),\sigma'_{\mathsf{pub}}) = 1$ (note that an honest signer issues a signature string only if the validity of a receiver public key is successfully verified). This means that adversary $\mathcal{A}$ must generate a key forgery as captured by the event $\mathsf{KeyForge}_\mathcal{A}$ in the proof of Theorem 4.2, and which can be defined and analyzed analogously here (with just minor syntactical changes).[4] Therefore, the event $\mathsf{SignForge}_\mathcal{A}$ is bounded by $\mathsf{KeyForge}$, i.e. $\Pr[\mathsf{SignForge}_\mathcal{A}] \leq \Pr[\mathsf{KeyForge}_\mathcal{A}]$, and the analysis of event $\mathsf{KeyForge}_\mathcal{A}$ follows the same reasoning as in Lemma 4.3 to conclude that $\Pr[\mathsf{KeyForge}_\mathcal{A}] \leq \mathsf{Adv}^{\mathrm{SUF\text{-}CMA}}_{\mathsf{DS_{pub}},\mathcal{B}_0}(\lambda)$. This results in the fact that $\Pr[\mathsf{SignForge}_\mathcal{A}] \leq \mathsf{Adv}^{\mathrm{SUF\text{-}CMA}}_{\mathsf{DS_{pub}},\mathcal{B}_0}(\lambda)$, which proves the lemma. □

Lemma 4.9 (Transition from G_1 to G_2). *For any valid PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_1$ such that*

$$|\Pr[\mathsf{G}_1(\lambda,\mathcal{A})=1] - \Pr[\mathsf{G}_2(\lambda,\mathcal{A})=1]| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda).$$

[4] The event is essentially identical with only syntactical adjustments since the random experiment changes. The precise definition would be that the adversary $\mathcal{A}$ terminates and there exists an entry $(\cdot,\mathsf{pk}^*_S,\mathsf{pk}^*_R,m^*,\sigma^*) \in \mathcal{QS}$ for which the following condition holds: $\mathsf{Verify}(\mathsf{mpk},\mathsf{pk}^*_S,\mathsf{pk}^*_R,m^*,\sigma^*) = 1 \wedge ((\cdot,\mathsf{pk}^*_S,\cdot,\cdot,\cdot) \notin \mathcal{QK} \vee (\cdot,\mathsf{pk}^*_R,\cdot,\cdot,\cdot) \notin \mathcal{QK})$. The reduction works as in the proof of Lemma 4.3, i.e., adversary $\mathcal{B}_0$ emulates the game and uses the signing oracle of $\mathsf{DS_{pub}}$ when issuing PCS public keys. When adversary $\mathcal{A}$ terminates the above event implies a forgery.

Proof. We build an adversary $\mathcal{B}_1$ that simulates $\mathsf{G}_{1+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\mathrm{ZK}_\beta^{\mathsf{NIZK}}$ experiment.

In the beginning of the reduction, $\mathcal{B}_1$ receives F from adversary $\mathcal{A}$ and CRS from the $\mathrm{ZK}_\beta^{\mathsf{NIZK}}$ experiment. It generates two signature key pairs $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^\lambda)$ and $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$, a predicate encryption master secret key $\mathsf{msk}_{\mathsf{PE}} \leftarrow \mathsf{PE}.\mathsf{Setup}(1^\lambda)$, sets $(\mathsf{mpk}, \mathsf{msk}) = ((F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}), (\mathsf{msk}_{\mathsf{PE}}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}))$ and gives mpk to the adversary $\mathcal{A}$. The adversary $\mathcal{B}_1$ also initializes the lists $\mathcal{QK} = \{\}$ and $\mathcal{QC} = \{\}$ and the counter $i = 1$.

For every left-or-right key generation query (x_0, x_1) asked by $\mathcal{A}$, $\mathcal{B}_1$ generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x_0)$ and sends pk as a reply to $\mathcal{A}$. Additionally, it adds $(i, \mathsf{pk}, \mathsf{sk}, x_0, x_1)$ to $\mathcal{QK}$ and increases the counter i, i.e. $i := i + 1$.

Whenever $\mathcal{A}$ asks a corruption query using the index j to the corruption oracle QCor, the adversary $\mathcal{B}_1$ checks the list $\mathcal{QK}$ to find $(j, \mathsf{pk}, \mathsf{sk}, x_0, x_1)$. If no such entry exists, the adversary $\mathcal{B}_1$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk to $\mathcal{A}$ and adds $(j, \mathsf{pk}, \mathsf{sk}, x_0, x_1)$ to $\mathcal{QC}$.[5]

For every sign query (j, pk_R, m) to QSign' asked by $\mathcal{A}$, $\mathcal{B}_1$ checks the list $\mathcal{QK}$ to find $(j, \mathsf{pk}, \mathsf{sk}, x_S^0, x_S^1)$ and $(\cdot, \mathsf{pk}_R, \cdot, x_R^0, x_R^1)$, parses $\mathsf{pk}_R = (\mathsf{vk}_R, \mathsf{ct}_R, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} = (\mathsf{vk}_S, \mathsf{sk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$. If no entry $(j, \mathsf{pk}, \mathsf{sk}, x_S^0, x_S^1)$ or $(\cdot, \mathsf{pk}_R, \cdot, x_R^0, x_R^1)$ is contained in the list $\mathcal{QK}$ or if $F(x_S^0, x_R^0) = 0$, then the adversary $\mathcal{B}_1$ outputs $\perp$.[6] Otherwise, $\mathcal{B}_1$ submits $((\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_S, \mathsf{ct}_R), (\mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}}))$ to the prove oracle which replies with π. The adversary $\mathcal{B}_1$ finally produces the signature $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{P}}, (m, \mathsf{pk}_R, \pi))$ and returns $\sigma := (\pi, \sigma')$ to $\mathcal{A}$.

Finally, the adversary $\mathcal{B}_1$ outputs the same bit β' returned by $\mathcal{A}$. To conclude the proof, we observe that our emulation is perfect. This follows from the fact that the only difference in the two games is the generation of the CRS and the proofs contained in the signatures, which is done by the underlying challenger. In the case that the challenger outputs an honestly generated CRS and honestly generated proofs, the adversary $\mathcal{B}_1$ is simulating the game G_1 and in the case that the challenger simulates the CRS and the proofs, the adversary $\mathcal{B}_1$ is simulating the game G_2. Note that by the perfect correctness of the predicate encryption scheme, we know that the challenger always replies, i.e., we have that $\mathsf{PE}.\mathsf{Dec}(\mathsf{sk}_{f_{\boldsymbol{x}}}, \mathsf{ct}_R) = F(x_S^0, x_R^0)$ (since all PCS keys can be assumed to be honestly generated in this hybrid experiment), and thus we are in fact submitting a valid witness and if the policy is not satisfied, returning $\perp$ is the correct behavior. This covers the simulation of the game $\mathsf{G}_{1+\beta}$ and leads to the advantage mentioned in the lemma. ☐

Lemma 4.10 (Transition from G_2 to G_3). *For any valid PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_2$ such that*

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

Proof. We build an adversary $\mathcal{B}_2$ that simulates $\mathsf{G}_{1+\beta}$ to $\mathcal{A}$ when interacting with the underlying $\mathrm{AH}_\beta^{\mathsf{PE}}$ experiment.

In the beginning of the reduction, $\mathcal{B}_2$ receives F from the adversary $\mathcal{A}$. It simulates a CRS, i.e., $(\mathsf{CRS}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$, generates two signature key pairs $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^\lambda)$ and $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$, sets $\mathsf{mpk} := (F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}})$ and sends mpk to $\mathcal{A}$. The adversary $\mathcal{B}_2$ also initializes the lists $\mathcal{QK} = \{\}$ and $\mathcal{QC} = \{\}$ and the counter $i = 1$.

For every left-or-right key generation query (x_0, x_1) asked by $\mathcal{A}$, $\mathcal{B}_2$ checks that $F(x_0', x_0) = F(x_1', x_1)$ for all $(\cdot, \cdot, \cdot, x_0', x_1') \in \mathcal{QC}$. If this check is unsuccessful, the adversary $\mathcal{B}_2$ outputs a random bit $\alpha \leftarrow \{0, 1\}$ as its guess and aborts.[7] If the check is successful, the adversary $\mathcal{B}_2$ computes

[5] We note in passing that for a valid PCS adversary, it must hold that $x_0 = x_1$.

[6] We again note in passing that for a valid adversary, it must hold that $F(x_S^0, x_R^0) = F(x_S^1, x_R^1)$.

[7] Looking ahead, the output of a random bit happens in cases where the adversary is not valid (cf. Definition 3.3) and thus we do not lose any advantage.

$(f_{\boldsymbol{x}_0}, \boldsymbol{x}_0) = \mathsf{SubPol}(F, x_0)$ and $(f_{\boldsymbol{x}_1}, \boldsymbol{x}_1) = \mathsf{SubPol}(F, x_1)$ and submits the left-or-right challenge query $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ to its own oracle to receive ct as a reply. In the next step, $\mathcal{B}_2$ samples a digital signature key pair $(\mathsf{vk}_\mathsf{P}, \mathsf{sk}_\mathsf{P}) \leftarrow \mathsf{DS}_\mathsf{P}.\mathsf{Setup}(1^\lambda)$, computes $\sigma_\mathsf{pub} \leftarrow \mathsf{DS}_\mathsf{pub}.\mathsf{Sign}(\mathsf{sk}_\mathsf{pub}, (\mathsf{vk}_\mathsf{P}, \mathsf{ct}))$ and outputs $\mathsf{pk} = (\mathsf{vk}_\mathsf{P}, \mathsf{ct}, \sigma_\mathsf{pub})$ to $\mathcal{A}$. Additionally, $\mathcal{B}_2$ adds $(i, \mathsf{pk}, \mathsf{sk}_\mathsf{P}, x_0, x_1)$ to $\mathcal{QK}$ and increases the counter i, i.e. $i := i + 1$.[8]

Whenever $\mathcal{A}$ asks a corruption query j, $\mathcal{B}_2$ checks the list $\mathcal{QK}$ to find $(j, \mathsf{pk}, \mathsf{sk}_\mathsf{P}, x_0, x_1)$. If no such entry exists, the adversary $\mathcal{B}_2$ returns $\perp$. If it holds that $x_0 \neq x_1$ or if $F(x_0, x_0') \neq F(x_1, x_1')$ for any $(\cdot, \cdot, \cdot, x_0', x_1') \in \mathcal{QK}$, then the adversary $\mathcal{B}_2$ outputs a random bit $\alpha \leftarrow \{0, 1\}$ as its guess and aborts (again, this only occurs for an invalid $\mathcal{A}$). Otherwise, $\mathcal{B}_2$ computes $(f_{\boldsymbol{x}_0}, \boldsymbol{x}_0) = \mathsf{SubPol}(F, x_0)$, submits the key generation query $f_{\boldsymbol{x}_0}$ to its challenger and receives as a reply $\mathsf{sk}_{f_{\boldsymbol{x}_0}}$. In the next step, $\mathcal{B}_2$ computes $\sigma_\mathsf{priv} \leftarrow \mathsf{DS}_\mathsf{priv}.\mathsf{Sign}(\mathsf{sk}_\mathsf{priv}, (\mathsf{vk}_\mathsf{P}, \mathsf{sk}_{f_{\boldsymbol{x}_0}}))$ and outputs $\mathsf{sk} = (\mathsf{vk}_\mathsf{P}, \mathsf{sk}_\mathsf{P}, \mathsf{sk}_{f_{\boldsymbol{x}_0}}, \sigma_\mathsf{priv})$ to $\mathcal{A}$, where sk_P is the signature key generated during the key generation oracle query contained in $\mathcal{QK}$. Additionally, $\mathcal{B}_2$ adds $(j, \mathsf{pk}, \mathsf{sk}, x_0, x_1)$ to $\mathcal{QC}$.

For every sign query (j, pk_R, m) to QSign' asked by $\mathcal{A}$, $\mathcal{B}_2$ checks the list $\mathcal{QK}$ to find $(j, \mathsf{pk}, \mathsf{sk}_\mathsf{P}, x_0, x_1)$ and $(\cdot, \mathsf{pk}_R, \cdot, \cdot)$. If no entry $(j, \mathsf{pk}, \mathsf{sk}_\mathsf{P}, x_0, x_1)$ or $(\cdot, \mathsf{pk}_R, \cdot, \cdot)$ is contained in the list $\mathcal{QK}$, then the adversary $\mathcal{B}_2$ outputs $\perp$. Otherwise, $\mathcal{B}_2$ checks, in the next step, that the attributes associated with the public keys pk_S and pk_R fulfill the policy, i.e. it checks that $F(x_S^0, x_R^0) = 1$ and $F(x_S^1, x_R^1) = 1$, with $(j, \cdot, \cdot, x_S^0, x_S^1) \in \mathcal{QK}$ and $(\cdot, \mathsf{pk}_R, \cdot, x_R^0, x_R^1) \in \mathcal{QK}$. If this is the case, the adversary $\mathcal{B}_2$ simulates the proof π for the language L (defined in Fig. 9),[9] generates the signature $\sigma' \leftarrow \mathsf{DS}_\mathsf{P}.\mathsf{Sign}(\mathsf{sk}_\mathsf{P}, (m, \mathsf{pk}_R, \pi))$ and outputs $\sigma = (\pi, \sigma')$ to $\mathcal{A}$. If the attributes associated with the public keys pk_S and pk_R do not fulfill the policy, i.e. $F(x_S^0, x_R^0) = 0$ or $F(x_S^1, x_R^1) = 0$, the adversary $\mathcal{B}_2$ outputs $\perp$ to $\mathcal{A}$. In the case that the policy evaluations differ, i.e. it holds that either $F(x_S^0, x_R^0) = 0$ and $F(x_S^1, x_R^1) = 1$ or $F(x_S^0, x_R^0) = 1$ and $F(x_S^1, x_R^1) = 0$, $\mathcal{B}_2$ aborts and outputting a random bit $\alpha \leftarrow \{0, 1\}$ as its guess to the underlying challenger (the adversary $\mathcal{A}$ is invalid).

As the last step, $\mathcal{B}_2$ outputs the same bit β' returned by $\mathcal{A}$.

We need to argue that the adversary $\mathcal{B}_2$ is a valid adversary with respect to the $\mathrm{AH}_\beta^\mathsf{PE}$ experiment if the adversary $\mathcal{A}$ fulfills all the checks described above, i.e. is a valid adversary in the $\mathsf{G}_{2+\beta}$ ($\mathrm{AH}_\beta^\mathsf{PCS}$) game. One of the validity requirements above (and in Definition 3.3) that $\mathcal{A}$ needs to fulfill is that for every x where $x := x_0 = x_1$ with $(\cdot, \cdot, \cdot, x_0, x_1) \in \mathcal{QC}$ it needs to hold that $F(x, x_0) = F(x, x_1)$ for all the challenge queries (x_0, x_1). This results in the fact that $f_{\boldsymbol{x}}(\boldsymbol{x}_0) = f_{\boldsymbol{x}}(\boldsymbol{x}_1)$ with $(f_{\boldsymbol{x}}, \cdot) = \mathsf{SubPol}(F, x)$ for all x and with $(\cdot, \boldsymbol{x}_b) = \mathsf{SubPol}(F, x_b)$ for $b \in \{0, 1\}$, where (x_0, x_1) are all the challenge queries. This matches exactly the validity requirements asked for $\mathcal{B}_2$ in the $\mathrm{AH}_\beta^\mathsf{PE}$ experiment. Therefore, it follows that the adversary $\mathcal{B}_2$ is a valid adversary with respect to the $\mathrm{AH}_\beta^\mathsf{PE}$ experiment and does not abort if the adversary $\mathcal{A}$ is a valid adversary in the game $\mathsf{G}_{2+\beta}$ ($\mathrm{AH}_\beta^\mathsf{PCS}$).

To conclude the proof, we observe that the difference in the two games is the generation of the challenge public keys pk, which either consists of a ciphertext encrypting the attribute set x_0 or the attribute set x_1. The computation of the ciphertexts is done by the underlying challenger of the attribute-hiding game. Together with the analysis above, it follows that, for a valid adversary $\mathcal{A}$, the game $\mathsf{G}_{2+\beta}$ is simulated towards $\mathcal{A}$ when the challenger encrypts the attribute set x_β for $\beta \in \{0, 1\}$. This concludes the simulation of the game $\mathsf{G}_{2+\beta}$ and the lemma follows. $\square$

[8] Note that it is vital not to obtain the PE functional keys of honest parties. Since we simulate proofs, we also do not need them. The only information we need to remember in this step is the key sk_P to sign the final message.

[9] The check regarding the associated policies together with the correctness of the PE scheme ensure that the statement for which we simulate the proof is in the language L (and thus, we do not need to rely on additional properties of the NIZK such as simulation soundness).

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 [KSW08, BW07]. The two functionality classes we recall are:

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

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

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 $F^{\mathrm{IP}}_{N,k}(\boldsymbol{x}, \cdot)$ and $F^{\mathrm{HV}}_k(\boldsymbol{x}, \cdot)$ parameterized by the vector $\boldsymbol{x}$ corresponding to the first argument of the above functions, respectively. As shown in [KSW08], 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 [OT12b, OT12a] (and also for a certain type of simulation-based security [DOT18]). 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 [BLS01] (or BB signatures [BB04] 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 $R_{\mathrm{ZK}}(x, w) \leftrightarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}_S, \mathsf{sk}_{f_{\boldsymbol{x}}}), \sigma_{\mathsf{priv}}) \wedge \mathsf{Dec}(\mathsf{sk}_{f_{\boldsymbol{x}}}, \mathsf{ct}_R) = 1$, 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 [GS08, EG14] 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. [KPW15] in combination with the (pairing-based) inner-product PE scheme from Okamoto et al. [OT12a] 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 pairing product 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 pairing product equations are only witness-indistinguishable unless certain additional constraints are met [GS08]. 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 [CKLM12a] 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 [CKLM12b, Theorem 3.2 and Appendix B] for the full details. As mentioned above, we instantiate the paring-based primitives from [OT12a] (encryption) and [KPW15] (signature):

– In the PE scheme of [OT12a], 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 $c_1 = (A_1, \dots A_N)$ 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 $\mathsf{sk}_{f_{\boldsymbol{x}}} = (k_1, \dots, k_N)$ 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 [KPW15], 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 $m := (g_1, x_1, \dots, x_n)$: 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. 9 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 [KSW08] 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. 8, 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 $f_{\boldsymbol{x}}(\cdot) := F^{\mathrm{IP}}_{N,k+1}((X_1, \dots, X_k, 1), \cdot)$, 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 $\boldsymbol{x} := (Y_1, \dots, Y_k, -d)$.

We observe that if a sender obtains a secret key generated based on the vector $(X_1, \dots, X_k, 1)$ and signs (as shown in Fig. 8) a message for a receiver public key that contains the ciphertext generated based on vector $(Y_1, \dots, Y_k, -d)$ as shown above, we have

$$\langle (X_1, \dots, X_k, 1), (Y_1, \dots, Y_k, -d) \rangle = 0 \iff \langle (X_1, \dots, X_k), (Y_1, \dots Y_k) \rangle = d$$

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 $i \in [k]$

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

(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 $i \in [k]$

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

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. 8). 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 [KSW08, BW07]. 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. 8.

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 [Can03, BH04] that incorporates all of our declared goals for this primitive. 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 parameter 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 $(\textsc{verify}, \mathrm{sid}, m, \sigma, v'_M, v'_A, v'_B)$, 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 (and with respect to public parameter v_M) unless the public keys are associated to attributes x'_A and x'_B such that $F(x'_A, x'_B) = 1$.
– 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 $F(\tilde{x}, x_j)$.

The functionality is specified below:

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

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

Policy Initialization. Upon input $(\textsc{policy-gen}, \mathrm{sid}, F)$ from party M do the following: if $v_M \neq \perp$ or $F \notin \mathcal{F}$, ignore the request; otherwise, provide $(\textsc{policy-gen}, \mathrm{sid}, F)$ to $\mathcal{S}$. Upon receiving $(\textsc{policy-gen}, \mathrm{sid}, v)$ from $\mathcal{S}$, verify that no entry $(\cdot, \cdot, v, \cdot, \cdot, 1)$ is recorded and ignore the reply if there is such an entry. Else, set $v_M \leftarrow v$ and output $(\textsc{policy-gen}, \mathrm{sid}, v)$ to M.

Key Generation. Upon input $(\textsc{key-gen}, \mathrm{sid}, P, x)$ from party M, where $P \in \mathcal{P} \setminus \mathcal{I}$, do the following: ignore the request if $v_M = \perp$; otherwise define $x_P \leftarrow x$ and compute:

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

Signing. On input $(\textsc{sign}, \mathrm{sid}, m, v)$ 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 $(\textsc{sign}, \mathrm{sid}, m, P, v, 1)$ to $\mathcal{S}$. Upon receiving $(\textsc{signature}, \mathrm{sid}, m, P, v, \sigma)$ from $\mathcal{S}$, verify that no entry $(m, \sigma, v_M, v_P, v_{P_j}, 0)$ is stored and ignore the reply if there is such an entry. Else, output $(\textsc{signature}, \mathrm{sid}, m, v, \sigma)$ to P, and store the entry $(m, \sigma, v_M, v_P, v_{P_j}, 1)$.
– In any other case, provide $(\textsc{sign}, \mathrm{sid}, m, P, v, 0)$ to $\mathcal{S}$ and when receiving $(\textsc{signature}, \mathrm{sid}, m, s)$ from $\mathcal{S}$, output $(\textsc{signature}, \mathrm{sid}, m, v, s)$ to P. *(This case guarantees that such messages are not considered as signed.)*

Verification. Upon input $(\textsc{verify}, \mathrm{sid}, m, \sigma, v'_M, v'_A, v'_B)$ from any party P, hand $(\textsc{verify}, \mathrm{sid}, m, \sigma, v'_M, v'_A, v'_B)$ to $\mathcal{S}$. Upon receiving $(\textsc{verified}, \mathrm{sid}, m, v'_M, v'_A, v'_B, \phi)$ 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 $P_i, P_j \in \mathcal{I}$, then set $f = 0$ in case $F(x_{P_i}, x_{P_j}) = 0$, 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 $(m, \sigma, v'_M, v'_A, v'_B, f')$ 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 $(\textsc{verified}, \mathrm{sid}, m, f)$ to P.

Corruption Mode. The party M is not corruptible. For all other parties P_i, the functionality supports the standard corruption mode [Can20], that is, upon input $(\textsc{corrupt}, P_i)$ 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 [Can03, BH04], 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 $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$ (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[10], where we assume secure point-to-point channels between M and each P_i. The protocol π^{PCS}_M 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 $(\mathsf{mpk}, \mathsf{msk})$ internally.
  - On input $(\textsc{key-gen}, \mathrm{sid}, P, x)$, run $\mathsf{KeyGen}(\mathsf{msk}, x)$ and send the message $(x, \mathsf{mpk}, (\mathsf{pk}, \mathsf{sk}))$ to party P over a secure channel.

– **Party P_i:**
  - Upon receiving (for the first time) the message $(x, \mathsf{mpk}, (\mathsf{pk}, \mathsf{sk}))$ 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 $\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, v, m)$ 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 [BSW11, KLM+18, MM15] that implies the UC

[10] 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.

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 [BSW11,KLM+18,MM15], we consider a simulator $\mathcal{S} = (\mathcal{S}_{\text{Setup}}, \mathcal{S}_{\text{KG}}, \mathcal{S}_{\text{Cor}}, \mathcal{S}_{\text{Sgn}})$, where $\mathcal{S}_{\text{Setup}}$ simulates Setup and $\mathcal{S}_{\text{KG}}$, $\mathcal{S}_{\text{Cor}}$, and $\mathcal{S}_{\text{Sgn}}$ 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* $\text{Real}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ *and* $\text{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S})$ *for a PPT adversary* $\mathcal{A}$ *and a PPT simulator* $\mathcal{S} = (\mathcal{S}_{\text{Setup}}, \mathcal{S}_{\text{KG}}, \mathcal{S}_{\text{Cor}}, \mathcal{S}_{\text{Sgn}})$ *in Fig. 10. In the real world, the adversary 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* $\mathcal{L}$*. Furthermore, the ideal key-generation oracle is formally implemented by the function* $\mathcal{S}'_{\text{KG}}(s, \mathcal{L}, x) := \mathcal{S}_{\text{KG}}(s, \mathcal{L})$*. The leakage set is initially empty. The sets are updated according to the following rules:*

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

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

After the simulator $\mathcal{S}_{\text{KG}}$ *has been invoked,* (j, pk_j, x_j) *is added to* $\mathcal{IQK}$*, where* pk_j *is the output of* $\mathcal{S}_{\text{KG}}$*.*

– *When* $\mathcal{A}$ *makes a corruption query* i *with* $(i, \mathsf{pk}_i, x_i) \in \mathcal{IQK}$*, then the following gets added to* $\mathcal{L}$ *(before* $\mathcal{S}_{\text{Cor}}$ *is invoked):*

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

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:*

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

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

$\mathbf{Real}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$	$\mathbf{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S})$
$(F, \tau) \leftarrow \mathcal{A}_1(1^\lambda)$	$(F, \tau) \leftarrow \mathcal{A}_1(1^\lambda)$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, F)$	$(\mathsf{mpk}, s) \leftarrow \mathcal{S}_{\mathrm{Setup}}(1^\lambda, F)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QKeyGen}(\cdot),\mathsf{QCor}(\cdot),\mathsf{QSign}(\cdot,\cdot,\cdot)}(\tau, \mathsf{mpk})$	$\alpha \leftarrow \mathcal{A}^{\mathcal{S}'_{\mathrm{KG}}(s,\mathcal{L},\cdot),\mathcal{S}_{\mathrm{Cor}}(s,\mathcal{L},\cdot),\mathcal{S}_{\mathrm{Sgn}}(s,\mathcal{L},\cdot,\cdot,\cdot)}(\tau, \mathsf{mpk})$
Output: α	**Output:** α

Fig. 10: 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 interacts with a simulator $\mathcal{S} = (\mathcal{S}_{\mathrm{Setup}}, \mathcal{S}_{\mathrm{KG}}, \mathcal{S}_{\mathrm{Cor}}, \mathcal{S}_{\mathrm{Sgn}})$. The simulator gets the initially empty leakage set $\mathcal{L}$, which grows during the experiment as described in Definition 5.1 to answer the oracle queries.

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. 6. 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 $\mathcal{S}_{pcs} = (\mathcal{S}_{\mathrm{Setup}}, \mathcal{S}_{\mathrm{KG}}, \mathcal{S}_{\mathrm{Cor}}, \mathcal{S}_{\mathrm{Sgn}})$ 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 $(\textsc{key-gen}, \mathrm{sid}, P, x)$, 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 [Can03] with additional setup, that is. We define the simulator $\mathcal{S}_{uc}$:

- On input $(\textsc{policy-gen}, \mathrm{sid}, F)$ from Func_{hyb}, execute $\mathsf{Setup}(1^\lambda, F)$ and then return to the functionality $(\textsc{policy-gen}, \mathrm{sid}, \mathsf{mpk})$. Store msk for future use.
- On input $(\textsc{key-gen}, \mathrm{sid}, P_i)$ alongside the leakage set $\{(\hat{P}, P_i) \mapsto F(x_{\hat{P}}, x_{P_i}) \mid \text{for all corrupted } \hat{P} \in \mathcal{I}\}$, *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 $(\textsc{verification-key}, \mathrm{sid}, P_i, \mathsf{pk})$ to the functionality.
- On input $(\textsc{sign}, \mathrm{sid}, m, P, v, b)$ 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 $(\textsc{signature}, \mathrm{sid}, m, P, v, \sigma)$.
- On input $(\textsc{verify}, \mathrm{sid}, m, \sigma, v'_M, v'_A, v'_B)$ from Func_{hyb}, let $\phi \leftarrow \mathsf{Verify}(v'_M, v'_A, v'_B, m, \sigma)$ and return $(\textsc{verified}, \mathrm{sid}, m, v'_M, v'_A, v'_B, \phi)$ 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 $\{(P_i, P_j) \mapsto F(x_{P_i}, x_{P_j}) \mid P_j \in \mathcal{I}\}$ 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.)

Claim 5.3. *No UC environment $\mathcal{Z}$ can distinguish an execution of π_M^{PCS} (w.r.t. the dummy adversary) and an execution of the ideal protocol for functionality Func_{hyb} w.r.t. ideal adversary $\mathcal{S}_{uc}$.*

Proof. We define events on the UC execution to help establish the claim. In the following, we say a party P_i is initialized in the real execution if either (1) it at some point in time at which P_i is not corrupted, it produced output $(\text{VERIFICATION-KEY}, \text{sid}, P_i, ., .)$, or (2) at some point in time at which P_i is corrupted, M received the input $(\text{KEY-GEN}, \text{sid}, P, .)$. Clearly, when a party is considered initialized in the real world then by definition of the simulator, $P_i \in \mathcal{I}$ holds in the ideal world (w.r.t simulator $\mathcal{S}_{uc}$).

- Let B_0 be the event that upon the initialization of an honest party P_i, the computed key pair $(\mathsf{pk}, \mathsf{sk})$ (either real or simulated) is such that there exists a different party P' associated to key pair $(\mathsf{pk}', \mathsf{sk}')$ with $\mathsf{pk} = \mathsf{pk}'$.
- Let B_1 be the event that at some point in time at which a signing party P_i is not corrupted, $\mathcal{Z}$ requests any party to verify a valid signature σ of a message m w.r.t. P_i's public key and any receiver public key, where m has not been signed before by an explicit request of P_i.
- Let B_2 be the event that at some point in time $\mathcal{Z}$ requests any party to verify a valid signature σ of a message m w.r.t. the public key of some initialized sender party P_i and the public key of the initialized receiver P_j, but where $F(x_i, x_j) = 0$.
- Let B_3 be the event that at some point in time $\mathcal{Z}$ requests any party to verify a valid signature σ of a message m w.r.t. some public key v_S and some initialized receiver party P_j (corrupted or not), where v_S is not the result of an initialization for any party in this session.
- Let B_4 be the event that at some point in time $\mathcal{Z}$ requests any party to verify a valid signature σ of a message m w.r.t. the public key of some initialized sender party P_i (corrupted or not) and some public key v_R that is not the result of an initialization for any party in this session.
- Let B_5 be the event that the initialization of M results in a key v_M but there has been a previous evaluation $1 \leftarrow \mathsf{Verify}(v'_M, v'_A, v'_B, m, \sigma)$.

We argue that the real world and the ideal world with the above simulator do not differ until event $B := \bigvee_{i=0}^{5} B_i$ occurs. We observe that $\mathcal{S}_{uc}$ simulates the real world perfectly and is only restricted by Conditions 1 to 5 upon verification within Func_{hyb}, by the uniqueness requirement of parties' public keys upon key generation, and by the requirement that the key of the credential manager be initialized to a value w.r.t- which no successful evaluations exist yet. By definition and the correctness of the PCS scheme, Conditions 1 and 5 are satisfied. Events B_0 and B_5 can only occur with at most negligible probability: for B_0 this is straightforward as collisions of public keys upon key generation imply forgeries (cf. Section 3.2 and the general argument in the proof of Lemma 4.4). For B_5 it is obvious that the two conditions in Fig. 6 are not restricting, since no party has been initialized yet. It remains that the reason for an observable difference occurs in case a verification answer ϕ in $(\text{VERIFIED}, \text{sid}, m, v'_M, v'_A, v'_B, \phi)$ from $\mathcal{S}_{uc}$ to Func_{hyb} is changed to $f \neq \phi$ due to Conditions 2, 3, or 4. However, Condition 2 is captured by event B_1, Condition 3 is captured by event B_2, and Condition 4 is captured by event $B_3 \vee B_4$. The distinguishing advantage can thus be bounded by the probability of provoking B. In the remainder of this proof, we show how to obtain a PCS forger from any environment provoking each of the events B_i. Hence let $\mathcal{Z}$ be a UC environment and let us construct the respective forger G_i against $\mathbf{EUF\text{-}CMA}^{\mathsf{PCS}}$.

Case B_1**:** G_1 runs a simulation of $\mathcal{Z}$. When $\mathcal{Z}$ invokes the setup with policy F, G_1 outputs F to its challenger, and returns the obtained mpk. When $\mathcal{Z}$ request a key generation for party P_i, it generates the corresponding key via oracle $\mathsf{QKeyGen}$. When $\mathcal{Z}$ asks G_1 to sign a message m in the name of P_i for public key v, it generates the signature using oracle QSign. When $\mathcal{Z}$ corrupts a party P_i, obtain the private key via oracle QCor. When $\mathcal{Z}$ lets party P verify a signature σ for message m and w.r.t. master public key mpk, public (sender) key pk_i (from an uncorrupted and initialized party P_i) and some public (receiver) key pk_j (from an initialized party P_j), compute $f \leftarrow \mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_i, \mathsf{pk}_j, m, \sigma)$. If $f = 1$ and m was not requested to be signed for pk_R before by party P_i by the simulated $\mathcal{Z}$, then exit with forgery $(\mathsf{pk}_i, \mathsf{pk}_j, m, \sigma)$. Otherwise, answer the verification request with decision f. For any other verification request (for different master public keys), G_1 simply evaluates the algorithm Verify and returns the respective result.

The forgery output by G_1 is valid since m was never signed, P_i is honest and hence $(\mathsf{pk}_i, \cdot, \cdot) \notin \mathcal{QC}$. Therefore, the forgery accepts and is not trivial[11] according to experiment $\textbf{EUF-CMA}^{\mathsf{PCS}}$.

Case B_2**:** For this reduction the forger G_2 again internally runs a simulation of $\mathcal{Z}$ and uses its oracles to generate the public parameters. Different to above, after any key generation request by $\mathcal{Z}$ for some party P_i, G_2 obtains the private key sk_i via a corruption request to QCor. When $\mathcal{Z}$ asks G_2 to sign a message m in the name of some party P_i for public key v, G_2 generates the signature by evaluating $\sigma \leftarrow \mathsf{Sign}(v, \mathsf{sk}_i, m)$. The remaining steps in simulating the interaction with $\mathcal{Z}$ are done as above and G_2 behaves as follows when $\mathcal{Z}$ lets some party P verify a signature σ for message m and w.r.t. master public key mpk, public (sender) key pk_i and public (receiver) key pk_j: compute $f \leftarrow \mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_i, \mathsf{pk}_j, m, \sigma)$. If $f = 1$ and $f(x_i, x_j) = 0$ (note that since the parties are initialized, the associated attributes are known) then exit with forgery (m, σ). (As above, other verification requests are answered by just evaluating the request using Verify.)

The obtained forgery is valid, since $(\mathsf{pk}_i, \mathsf{pk}_j, m, \sigma)$ verifies and is not trivial, because G_2 does not use its signing oracle, and because $F(x_i, x_j) = 0$. Hence, the non-triviality conditions of $\textbf{EUF-CMA}^{\mathsf{PCS}}$ are met.

Case B_3**:** The forger G_3 simulates $\mathcal{Z}$ as above for G_1 above and behaves as follows when $\mathcal{Z}$ lets some party P verify a signature σ for message m and w.r.t. master public key mpk, public (sender) key pk and public (receiver) key pk_j: compute $f \leftarrow \mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}_j, m, \sigma)$. If $f = 1$ and pk is not a public key simulated towards $\mathcal{Z}$ as part of an initialization, then exit with forgery $(\mathsf{pk}, \mathsf{pk}_j, m, \sigma)$. (As above, other verification requests are answered by just evaluating the request using Verify.)

The obtained forgery is valid, since $(\mathsf{pk}, \mathsf{pk}_j, m, \sigma)$ verifies; it further cannot have been obtained from a signing oracle request because $(\mathsf{pk}_i, \cdot, \cdot) \notin \mathcal{QK}$ by definition of event B_3, and consequently, $(\mathsf{pk}_i, \cdot, \cdot) \notin \mathcal{QC}$ must hold. Hence, each part of the non-triviality condition of $\textbf{EUF-CMA}^{\mathsf{PCS}}$ is fulfilled.

Case B_4**:** For this case, the forger G_4 is defined as G_2 in terms of emulating the interaction with $\mathcal{Z}$, and G_4 behaves as follows when $\mathcal{Z}$ lets some party P verify a signature σ for message m and w.r.t. master public key mpk, public (sender) key pk_i and public (receiver) key pk: compute $f \leftarrow \mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_i, \mathsf{pk}, m, \sigma)$. If $f = 1$ and pk is not a public key simulated towards $\mathcal{Z}$ as part of an initialization, then exit with forgery $(\mathsf{pk}_i, \mathsf{pk}_j, m, \sigma)$. (As above, other verification requests are answered by just evaluating the request using Verify.)

[11] Recall that the forgery game is won if the following condition is satisfied: $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge (\mathsf{pk}, \mathsf{pk}^*, m^*, \cdot) \notin \mathcal{QS} \wedge \forall x, x^* : (\mathsf{pk}, \cdot, x) \in \mathcal{QC} \wedge (\mathsf{pk}^*, \cdot, x^*) \in \mathcal{QK} \Rightarrow F(x, x^*) = 0$.

The obtained forgery is valid, since $(\mathsf{pk}_i, \mathsf{pk}, m, \sigma)$ verifies and is not trivial, since G_4 does not use its signing oracle, and because $(\mathsf{pk}, \cdot, \cdot) \notin \mathcal{QK}$ (because pk cannot have been obtained from the key generation oracle).

We obtain a uniform forger G by choosing at random $i \in [4]$ and running G_i. This proves the claim. □

Hybrid H_2: This hybrid is the ideal functionality (i.e., the ideal protocol for) $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$ 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 $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$, 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 $L = \{(\hat{P}, P_i) \mapsto F(x_{\hat{P}}, x_{P_i}) \mid$ for all corrupted $\hat{P} \in \mathcal{I}\}$—*but without the additional leakage x from above*–from $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$, $\mathcal{S}'_{uc}$ 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 $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$, 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 $\sigma \leftarrow \mathcal{S}_{\text{Sgn}}(s, \mathcal{L}, P, v, m)$ (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 $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$, the simulator behaves identically to $\mathcal{S}_{uc}$.
- On a corruption request for party P_i, $\mathcal{S}'_{uc}$ corrupts P_i in $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$, and includes the additional leakage information $L = \{(P_i, x_{P_i})\} \cup \{(P_i, P_j) \mapsto F(x_{P_i}, x_{P_j}) \mid P_j \in \mathcal{I}\}$ 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 $\mathsf{sk} \leftarrow \mathcal{S}_{\text{Cor}}(s, \mathcal{L}, P_i)$ (which also updates s) and returns sk. (And from now, the simulator acts as relay between environment and functionality.)

Claim 5.4. *No UC environment $\mathcal{Z}$ can distinguish an execution of the ideal protocol for Func_{hyb} w.r.t. ideal adversary $\mathcal{S}_{uc}$ and an execution of the ideal protocol for functionality $\mathsf{Func}^{M,\mathcal{F}}_{\mathsf{PolSig}}$ w.r.t. ideal adversary $\mathcal{S}'_{uc}$.*

Proof (Sketch). Without loss of generality, we can assume an encoding of party identifiers (of the UC treatment) as integers (the identifiers in the simulation-based AH experiment). Then note that the leakage set maintained by $\mathcal{S}'_{uc}$ is exactly how the experiment $\mathbf{Ideal}^{\mathsf{PCS}}$ maintains the leakage sets for its simulators. We further observe that in the second hybrid world, adversary $\mathcal{S}_{uc}$ can be equivalently implemented using the oracles $\mathsf{QKeyGen}(\cdot)$, $\mathsf{QCor}(\cdot)$, and $\mathsf{QSign}(\cdot,\cdot,\cdot)$ to implement the key-generation, signing and corruption queries, and that $\mathcal{S}'_{uc}$ is structurally identical but implements those "oracle queries" for key-generation, signing, and corruption by calling the respective simulators $\mathcal{S}_{\text{KG}}$, $\mathcal{S}_{\text{Cor}}$, and $\mathcal{S}_{\text{Sgn}}$ using the same interface. That is, the systems H_1 and H_2 can be re-written as systems $S[\mathsf{Setup}, \mathsf{QKeyGen}, \mathsf{QCor}, \mathsf{QSign}]$ and $S[\mathcal{S}_{\text{Setup}}, \mathcal{S}_{\text{KG}}, \mathcal{S}_{\text{Cor}}, \mathcal{S}_{\text{Sgn}}]$, respectively, where $S[R_1, R_2, R_3, R_4]$ is a black box algorithm with respect to invocations of its subroutines R_i. Therefore, any environment $\mathcal{Z}$ with non-negligible advantage in distinguishing H_1 and H_2 contradicts the assumption that systems $\mathbf{Real}^{\mathsf{PCS}}$ and $\mathbf{Ideal}^{\mathsf{PCS}}$ are indistinguishable: the reduction $\mathcal{A}$ is obtained by internally emulating an instance of $\mathcal{Z}$ and running algorithm S which is served with subroutine inputs from the corresponding functions in experiments $\mathbf{Real}^{\mathsf{PCS}}$ or $\mathbf{Ideal}^{\mathsf{PCS}}$, respectively. Finally, $\mathcal{A}$ outputs whatever $\mathcal{Z}$ outputs. This leads to a distinguisher $\mathcal{A}$ with the same distinguishing advantage as $\mathcal{Z}$ has when distinguishing 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 [Can01]) 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. 8 satisfies the strong simulation-based PE security notion as defined in Definition 2.10, then the generic scheme achieves the simulation-based and therefore the composable notion of PCS. We note that the requirement in Definition 2.10 is the adaptive (and thus stronger) version of what is proven so far in the literature, such as in [DOT18, OT12a]. 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 [KLM+18] or the random oracle model.

Theorem 5.5. *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 8, is simulation private. Namely, for any PPT adversary* $\mathcal{A}$ *there exist PPT adversaries* $\mathcal{B}, \mathcal{B}'$ *and* $\mathcal{B}''$*, such that:*

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

Proof. The simulator $\mathcal{S}$ for the proof of this theorem is described in Fig. 11. Informally, the simulator $\mathcal{S}$ uses the simulator $\mathcal{S}_{\mathrm{Enc}}$ of the predicate encryption scheme to generate the ciphertexts that are part of the public keys and the simulator $\mathcal{S}_{\mathrm{KG}}$ of the predicate encryption scheme to generate the functional keys that are part of the secret keys. To be able to do this, the simulator $\mathcal{S}$ learns, in every key generation query, the policy evaluation of all the corrupted keys acting as senders with the attribute set of the queried key, and, in the case of a corruption query, the simulator $\mathcal{S}$ learns the attribute set of the requested key as well as all the policy evaluations where this key acts as a sender. To answer 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 4.7, we will make use of the fact 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 as otherwise, we obtain a forgery for $\mathsf{DS_{pub}}$.

To show that the ideal world with the simulator $\mathcal{S}$ is indistinguishable from the real world, we use a hybrid argument that we describe below.

Hybrid H_0: This hybrid is defined as the real PCS experiment $\mathrm{Real}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$.

Hybrid H_1: In this hybrid, we change the behavior of the sign oracle QSign and define a modified sign oracle QSign' (similar to the transition in the IND-based case). The oracle QSign' is defined as QSign with the difference that it only answers queries for receiver keys that have previously been output by the key generation oracle $\mathsf{QKeyGen}$, i.e. for a query (i, pk', m) with $(i, \cdot, \cdot) \notin \mathcal{QK}$ or $(\cdot, \mathsf{pk}', \cdot) \notin \mathcal{QK}$ the sign oracle QSign' outputs $\perp$. The transition from H_0 to H_1 is justified by bounding the key-forgery event $\mathsf{KeyForge}_{\mathcal{A}}$ as in in Lemma 4.8, thus

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

Hybrid H_2: In this hybrid, we change from an honestly generated CRS and honestly generated proofs to a simulated CRS and simulated proofs. That is, we (again) modify the signing oracle, and additionally, on input (j, pk_R, m), first obtain attributes x_R to pk_R for which $F(x_j, x_R) = 1$. If no entry is found, we return $\perp$, and otherwise the proof is simulated using the trapdoor, the master public key, the signer verification key, and the receiver PE ciphertext.[12] We further perform a syntactical change: upon PCS-key generation, we do not generate the secret keys of honest parties (as we do not need them for signing any more); we only generate a secret keys upon corruption queries. The transition from H_1 to H_2 is justified by the zero-knowledge property of NIZK. Namely, in Lemma 5.6, we exhibit a PPT adversary $\mathcal{B}_1$ such that:

$$|\Pr[\mathsf{H}_1(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{H}_2(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda).$$

Hybrid H_3: In this hybrid, we change from honestly generated keys to simulated keys, that is, whenever we generate PE ciphertexts (as part of PCS key generation for attributes x), or PE functional keys (upon corruption queries), we maintain the leakage $\mathcal{L}_{\mathsf{PE}}$ by recording all the mappings

$$(i, j) \mapsto f_j(x_i) = F(x_j, x_i)$$

for all indices j for which a corruption query has been issued, and indices i for which a public key has been generated using attributes x_i. Upon every additional (say, the kth) PCS key-generation query with attributes x_k, the mappings $(k, j) \mapsto f_j(x_k) = F(x_j, x_k)$ are added (for all indices j in the corruption set). Upon every additional corruption query specifying index ℓ the mappings $(i, \ell) \mapsto f_\ell(x_i) = F(x_\ell, x_i)$ are added, where x_ℓ is the attribute corresponding to the ℓth PCS key-generation query. With this leakage set, the respective simulators $\mathcal{S}_{\mathrm{Enc}}$ and $\mathcal{S}_{\mathrm{KG}}$ are invoked, where $\mathcal{S}' = (\mathcal{S}_{\mathrm{Setup}}, \mathcal{S}_{\mathrm{Enc}}, \mathcal{S}_{\mathrm{KG}})$ is the assumed PE simulator. Clearly, we also simulate the PE master secret key using $\mathcal{S}_{\mathrm{Setup}}$ in H_3. The transition from H_2 to H_3 is justified by the simulation-based AH property of PE and we prove in Lemma 5.7 that

$$|\Pr[\mathsf{H}_2(\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{H}_3(\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{Sim}}_{\mathsf{PE},\mathcal{B}_2,\mathcal{S}'}(\lambda).$$

Hybrid H_4: This is the ideal PCS experiment $\mathrm{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S})$ for the simulator defined in Fig. 11. We show that

$$\Pr[\mathsf{H}_3(\lambda, \mathcal{A}) = 1] = \Pr[\mathrm{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S}) = 1].$$

This is straightforward to see since in both systems, H_3 and $\mathrm{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S})$, the keys (master public key, public keys and secret keys) as well as the signatures are completely simulated using the simulators of the underlying NIZK and PE scheme. To conclude the proof, we need to argue that the leakage provided to the PE simulator can be obtained from the leakage set $\mathcal{L}$ that the PCS simulator receives. Furthermore, when simulating NIZK proofs, another policy evaluation takes place and we need to show that also this can be implemented based on $\mathcal{L}$. To do that, we take a more detailed look at the leakage:

$$\begin{aligned}\mathcal{L} :=& \{(i, j) \mapsto F(x_i, x_j) \mid (i, \mathsf{pk}_i, x_i) \in \mathcal{IQC}, (j, \mathsf{pk}_j, x_j) \in \mathcal{IQK}\} \\ &\cup (i, x_i, \{(i, j) \mapsto F(x_i, x_j) \mid (j, \mathsf{pk}_j, x_j) \in \mathcal{IQK}\} \mid (i, \mathsf{pk}_i, x_i) \in \mathcal{IQC}) \\ &\cup \{(i, j) \mapsto F(x_i, x_j) \mid (i, \mathsf{pk}_i, x_i), (j, \mathsf{pk}_R, x_j) \in \mathcal{IQK} \text{ with } (i, \mathsf{pk}_R, \cdot) \text{ asked to } \mathsf{QSign}\}.\end{aligned}$$

From the description of the set, it becomes clear that $(i, j) \mapsto F(x_i, x_j)$ is exactly the information the simulator of the NIZK needs to simulate the proof that is part of the

[12] Note that by the first game hop, we can assume that we find the key, as otherwise the signing oracle immediately outputs $\perp$.

signature for the query $(i, \mathsf{pk}_R, \cdot)$. The leakage set $\mathcal{L}_{\mathsf{PE}}$ that is required by the simulator of the PE scheme to generate the ciphertexts, which are part of the public key, and the functional keys, which are part of the secret key, can be constructed in the following way:

$$\begin{aligned}&\{(i,j) \mapsto F(x_i, x_j) \mid (i, \mathsf{pk}_i, x_i) \in \mathcal{IQC}, (j, \mathsf{pk}_j, x_j) \in \mathcal{IQK}\}\\ =&\{(i,j) \mapsto f_i(x_j) \mid (i, \mathsf{pk}_i, x_i) \in \mathcal{IQC}, (j, \mathsf{pk}_j, x_j) \in \mathcal{IQK}\}\\ &\overset{(i,j)\mapsto(i',j')}{\Rightarrow} \{(i',j') \mapsto f_{i'}(x_{j'}) \mid i' \in [n_{\mathsf{KeyGen}}], j' \in [n_{\mathsf{Enc}}]\} =: \mathcal{L}_{\mathsf{PE}},\end{aligned}$$

where j' is defined as j but i' is defined based on the order when (party) index i is corrupted. In more detail, if the k'th corruption query, asked by the adversary, corresponds to the l'th key, then it holds that $i' = k$ and $i = l$. This matches the description of $\mathcal{L}_{\mathsf{PE}}$ presented in Definition 2.10 and concludes our argument.

This shows the indistinguishability of the real and ideal PCS experiments. □

```
$\underline{\mathcal{S}_{\mathrm{Setup}}(1^\lambda, F):}$
$(\mathsf{CRS}, s') \leftarrow \mathsf{NIZK}.\mathcal{S}_1(1^\lambda)$
$s'' \leftarrow \mathsf{PE}.\mathcal{S}_{\mathrm{Setup}}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$
$\mathsf{mpk} := (F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}})$
$\mathcal{QK} = \{\}$
$\mathcal{QC} = \{\}$
$i := 1$
$s := (s', s'', \mathcal{QK}, \mathcal{QC}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}})$
Return $(\mathsf{mpk}, s)$
```

```
$\underline{\mathcal{S}_{\mathrm{Cor}}(s, \mathcal{L}, j):}$
Parse $s := (s', s'', \mathcal{QK}, \mathcal{QC}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}})$
If $(j, \cdot, \cdot) \notin \mathcal{QK}$:
  Return $\perp$
Derive $\mathcal{L}_{\mathsf{PE}}$ from $\mathcal{L}$ as described above
$(f_{\boldsymbol{x}}, \boldsymbol{x}_j) = \mathsf{SubPol}(F, x_j)$
$\mathsf{sk}_{f_{\boldsymbol{x}}} \leftarrow \mathsf{PE}.\mathcal{S}_{\mathrm{KG}}(s'', \mathcal{L}_{\mathsf{PE}}, f_{\boldsymbol{x}})$
$\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}))$
$\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$
Add $(j, \mathsf{pk}, \mathsf{sk}, x_j)$ to $\mathcal{QC}$
Return $\mathsf{sk}$
```

```
$\underline{\mathcal{S}_{\mathrm{KG}}(s, \mathcal{L}):}$
Parse $s := (s', s'', \mathcal{QK}, \mathcal{QC}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}})$
$(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$
Derive $\mathcal{L}_{\mathsf{PE}}$ from $\mathcal{L}$ as described above
$\mathsf{ct} \leftarrow \mathsf{PE}.\mathcal{S}_{\mathrm{Enc}}(s'', \mathcal{L}_{\mathsf{PE}})$
$\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}))$
$\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, \sigma_{\mathsf{pub}})$
Add $(i, \mathsf{pk}, \mathsf{sk}_{\mathsf{P}})$ to $\mathcal{QK}$
Set $i := i + 1$
Return $\mathsf{pk}$
```

```
$\underline{\mathcal{S}_{\mathrm{Sgn}}(s, \mathcal{L}, j, \mathsf{pk}_R, m):}$
Parse $\mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, \sigma^R_{\mathsf{pub}})$,
      $s := (s', s'', \mathcal{QK}, \mathcal{QC}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}})$
If $(j, \cdot, \cdot) \in \mathcal{QK} \wedge \exists k : (k, \mathsf{pk}_R, \cdot) \in \mathcal{QK}$
                              $\wedge \{(j, k) \mapsto 1\} \in \mathcal{L}$:
  Parse the entry $(j, \mathsf{pk}_S := (\mathsf{vk}_S, \cdot, \cdot), \cdot) \in \mathcal{QK}$
  $\pi \leftarrow \mathsf{NIZK}.\mathcal{S}_2(s', (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_S, \mathsf{ct}_R))$
  $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{P}}, (m, \mathsf{pk}_R, \pi))$, with
            $(j, \cdot, \mathsf{sk}_{\mathsf{P}}) \in \mathcal{QK}$
  Return $\sigma := (\pi, \sigma')$
Else
  Return $\perp$
```

Fig. 11: Description of the simulators $\mathcal{S}_{\mathrm{Setup}}, \mathcal{S}_{\mathrm{KG}}, \mathcal{S}_{\mathrm{Cor}}, \mathcal{S}_{\mathrm{Sgn}}$ for the simulation-based attribute-hiding security proof of our PCS scheme.

Lemma 5.6 (Transition from H_1 to H_2). *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_1$ such that*

$$|\Pr[\mathsf{H}_1(\lambda,\mathcal{A})=1]-\Pr[\mathsf{H}_2(\lambda,\mathcal{A})=1]| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda).$$

Proof. We build an adversary $\mathcal{B}_1$ that simulates $\mathsf{H}_{1+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\mathrm{ZK}^{\mathsf{NIZK}}_{\beta}$ experiment. In the beginning of the reduction, $\mathcal{B}_1$ receives F from adversary $\mathcal{A}$ and CRS from the $\mathrm{ZK}^{\mathsf{NIZK}}_{\beta}$ experiment.

For the setup generation, the adversary $\mathcal{B}_1$ generates two signature key pairs $(\mathsf{vk}_{\mathsf{pub}},\mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^{\lambda})$ and $(\mathsf{vk}_{\mathsf{priv}},\mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^{\lambda})$, a PE master secret key $\mathsf{msk}_{\mathsf{PE}} \leftarrow \mathsf{PE}.\mathsf{Setup}(1^{\lambda})$, sets $(\mathsf{mpk},s)=((F,\mathsf{CRS},\mathsf{vk}_{\mathsf{pub}},\mathsf{vk}_{\mathsf{priv}}),(\mathsf{msk}_{\mathsf{PE}},\mathsf{sk}_{\mathsf{pub}},\mathsf{sk}_{\mathsf{priv}}))$ and gives mpk to the adversary $\mathcal{A}$. The adversary $\mathcal{B}_1$ also initializes the lists $\mathcal{QK}=\{\}$ and $\mathcal{QC}=\{\}$ and a counter $i:=1$.

For every query x to the key generation oracle, submitted by $\mathcal{A}$, $\mathcal{B}_1$ generates $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(s,x)$ and sends pk as a reply to $\mathcal{A}$. Additionally, $\mathcal{B}_1$ adds $(i,\mathsf{pk},\mathsf{sk},x)$ to $\mathcal{QK}$ and increases the counter, i.e. $i:=i+1$. For every query j to the corruption oracle, asked by $\mathcal{A}$, $\mathcal{B}_1$ searches $(j,\mathsf{pk},\mathsf{sk},x)$ in $\mathcal{QK}$ and outputs sk. Additionally, $\mathcal{B}_1$ adds $(j,\mathsf{pk},\mathsf{sk},x)$ to $\mathcal{QC}$. If pk is not contained in $\mathcal{QK}$, $\mathcal{B}_1$ outputs $\perp$.

For every query (j,pk_R,m) submitted to the signing oracle QSign' by $\mathcal{A}$, $\mathcal{B}_1$ parses $\mathsf{pk}_R = (\mathsf{vk}_R,\mathsf{ct}_R,\sigma^R_{\mathsf{pub}})$ and $s=(\mathsf{msk}_{\mathsf{PE}},\mathcal{QK},\mathcal{QC},\mathsf{sk}_{\mathsf{pub}},\mathsf{sk}_{\mathsf{priv}})$ and checks that $(j,\mathsf{pk}_S,\mathsf{sk}_S := ((\mathsf{vk}_S,\mathsf{sk}^S_{\mathsf{P}},\mathsf{sk}_{f_x},\sigma_{\mathsf{priv}}),x_S) \in \mathcal{QK}$. If it holds that $(j,\cdot,\cdot,\cdot) \notin \mathcal{QK}$ or $(\cdot,\mathsf{pk}_R,\cdot,x_R) \notin \mathcal{QK}$ for any x_R, then output $\perp$. Afterwards, the adversary $\mathcal{B}_1$ verifies the signature of pk_R, i.e. $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_R,\mathsf{ct}_R),\sigma^R_{\mathsf{pub}})$ and if $F(x_S,x_R)=1$ (for some x_R associated with pk_R), it submits the query $((\mathsf{vk}_{\mathsf{priv}},\mathsf{vk}_S,\mathsf{vk}_R,\mathsf{ct}_R),(\mathsf{sk}_{f_x},\sigma_{\mathsf{priv}}))$ to the prove oracle of its challenger and receives π as a reply. If the signature σ^R_{pub} does not verify or the decryption outputs 0, then $\mathcal{B}_1$ outputs $\perp$. The adversary $\mathcal{B}_1$ then computes $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}^S_{\mathsf{P}},(m,\mathsf{pk}_R,\pi))$ and returns $\sigma := (\pi,\sigma')$ to $\mathcal{A}$. Finally, $\mathcal{B}_1$ outputs the same bit β' returned by $\mathcal{A}$.

To conclude the proof, we observe that our emulation is perfect. This follows from the fact that the only difference in the two hybrids is the generation of the CRS and the proofs contained in the signatures, which is done by the underlying challenger. In the case that the challenger outputs an honestly generated CRS and genuinely generated proofs, the adversary $\mathcal{B}_1$ is simulating the hybrid H_1 and in the case that the challenger simulates the CRS and the proofs, the adversary $\mathcal{B}_1$ is simulating the hybrid H_2. Note that the challenger always replies: the perfect correctness of the predicate encryption scheme nd the fact that the signing oracle QSign' is only queried using previously generated keys, we have that $\mathsf{PE}.\mathsf{Dec}(\mathsf{sk}_{f_x},\mathsf{ct}_R)=F(x_S,x_R)$ with $(\cdot,\mathsf{pk}_R,\cdot,x_R) \in \mathcal{QK}$ and therefore it is guaranteed that the decryption corresponds to the correct policy evaluation and thus the witness is valid. This covers the simulation of the hybrids $\mathsf{H}_{1+\beta}$ and leads to the advantage mentioned in the lemma. □

Lemma 5.7 (Transition from H_2 to H_3). *For any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}_2$, such that*

$$|\Pr[\mathsf{H}_2(\lambda,\mathcal{A})=1]-\Pr[\mathsf{H}_3(\lambda,\mathcal{A})=1]| \leq \mathsf{Adv}^{\mathrm{Sim}}_{\mathsf{PE},\mathcal{B}_2,\mathcal{S}'}(\lambda).$$

Proof. We build an adversary $\mathcal{B}_2$ that simulates $\mathsf{H}_{2+\beta}$ to $\mathcal{A}$ when interacting with the underlying $\mathrm{Sim}^{\mathsf{PE}}$ experiment.

In the beginning of the reduction, $\mathcal{B}_2$ receives F from the adversary $\mathcal{A}$. It then executes the following steps: It simulates a CRS, i.e. $(\mathsf{CRS},s') \leftarrow \mathsf{NIZK}.\mathcal{S}_1(1^{\lambda})$, generates two signature key pairs $(\mathsf{vk}_{\mathsf{pub}},\mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^{\lambda})$ and $(\mathsf{vk}_{\mathsf{priv}},\mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^{\lambda})$, sets $(\mathsf{mpk},s):=$

$((F, \mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}), (s', \mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}))$ and sends mpk to $\mathcal{A}$. The adversary $\mathcal{B}_2$ also initializes the lists $\mathcal{QK} = \{\}$ and $\mathcal{QC} = \{\}$ and the counter $i := 1$.[13]

Whenever $\mathcal{A}$ asks the i'th query x to the key generation oracle, $\mathcal{B}_2$ generates a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$, computes $(f_{\boldsymbol{x}}, \boldsymbol{x}) = \mathsf{SubPol}(F, x)$ and submits x to its PE encryption oracle to receive ct. In the next step, it computes $\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}))$ and outputs $\mathsf{pk} = (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, \sigma_{\mathsf{pub}})$, which is then sent to $\mathcal{A}$. Additionally, $(i, \mathsf{pk}, \mathsf{sk}_{\mathsf{P}})$ is added to $\mathcal{QK}$ and we set $i := i + 1$.

For every query j to the corruption oracle, asked by $\mathcal{A}$, $\mathcal{B}_2$ checks that $(j, \cdot, \cdot) \in \mathcal{QK}$. If $(j, \cdot, \cdot) \notin \mathcal{QK}$ it does nothing. Otherwise, the reduction retrieves x_j from the jth key-generation query and computes $(f_{\boldsymbol{x}}, \boldsymbol{x}) = \mathsf{SubPol}(F, x_j)$ and submits $f_{\boldsymbol{x}}$ as a key query to the PE key-generation oracle, which replies with $\mathsf{sk}_{f_{\boldsymbol{x}}}$. The simulator then computes $\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}))$ with $(\mathsf{pk} = (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, \sigma_{\mathsf{pub}}), \mathsf{sk}_{\mathsf{P}}, \cdot) \in \mathcal{QK}$, sets $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{sk}_{f_{\boldsymbol{x}}}, \sigma_{\mathsf{priv}})$, adds $(j, \mathsf{pk}, \mathsf{sk}, x_j)$ to $\mathcal{QC}$ and outputs sk to $\mathcal{A}$.

For every query (j, pk_R, m) asked by $\mathcal{A}$ to the signing oracle QSign', $\mathcal{B}$ checks that $(j, \cdot, \cdot) \in \mathcal{QK}$. If $(j, \cdot, \cdot) \notin \mathcal{QK}$ it does nothing, otherwise it searches for $(j, \mathsf{pk}_S = (\mathsf{vk}_S, \mathsf{ct}_S, \sigma^S_{\mathsf{pub}}), \mathsf{sk}_{\mathsf{P}}) \in \mathcal{QK}$ and parses $\mathsf{pk}_R = (\mathsf{vk}_R, \mathsf{ct}_R, \sigma^R_{\mathsf{pub}})$. If it cannot find $(j, \cdot, \cdot) \in \mathcal{QK}$ it outputs $\perp$, otherwise it verifies the signature of pk_R, i.e. $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_R, \mathsf{ct}_R), \sigma^R_{\mathsf{pub}})$ and if $F(x_S, x_R) = 1$ (where x_S are the attributes of the key associated with j and x_R are the attributes of the key associated with pk_R) it generates $\pi \leftarrow \mathsf{NIZK}.\mathcal{S}_2(s'', (\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_S, \mathsf{vk}_R, \mathsf{ct}_R))$. If the signature σ^R_{pub} does not verify or if $F(x_S, x_R) = 0$, it outputs $\perp$ to $\mathcal{A}$. The simulator then computes $\sigma' \leftarrow \mathsf{DS}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{P}}, (m, \mathsf{pk}_R, \pi))$ and returns $\sigma := (\pi, \sigma')$ to $\mathcal{A}$.

Finally, $\mathcal{B}_2$ outputs the same bit β' returned by $\mathcal{A}$.

To conclude the proof, we observe that our simulation is perfect. This follows from the fact that the only difference in the two hybrid is the generation of the public keys pk and secret keys sk, which either consist of honestly generated ciphertexts or functional keys or simulated ciphertexts or functional keys. The generation of the ciphertexts and functional keys is done by the underlying challenger of the simulation-based attribute hiding notion. In the case that the challenger generates an honest encryption and honest keys, the adversary $\mathcal{B}_2$ is simulating the hybrid H_1 and in the case that the challenger simulates ciphertexts and keys, the adversary $\mathcal{B}_2$ is simulating the hybrid H_2. □

References

AFNV19. G. Ateniese, D. Francati, D. Nuñez, and D. Venturi. Match me if you can: Matchmaking encryption and its applications. In *CRYPTO 2019, Part II*, *LNCS* 11693, pages 701–731. Springer, Heidelberg, August 2019. (Page 7.)

BB04. D. Boneh and X. Boyen. Short signatures without random oracles. In *EUROCRYPT 2004*, *LNCS* 3027, pages 56–73. Springer, Heidelberg, May 2004. (Page 28.)

BF14. M. Bellare and G. Fuchsbauer. Policy-based signatures. In *PKC 2014*, *LNCS* 8383, pages 520–537. Springer, Heidelberg, March 2014. (Pages 3 and 6.)

BGG$^+$90. M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In *CRYPTO'88*, *LNCS* 403, pages 37–56. Springer, Heidelberg, August 1990. (Page 9.)

BH04. M. Backes and D. Hofheinz. How to break and repair a universally composable signature functionality. In *ISC 2004*, *LNCS* 3225, pages 61–72. Springer, Heidelberg, September 2004. (Pages 6, 31, and 33.)

BLS01. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *ASIACRYPT 2001*, *LNCS* 2248, pages 514–532. Springer, Heidelberg, December 2001. (Page 28.)

BMM17. C. Badertscher, C. Matt, and U. Maurer. Strengthening access control encryption. In *ASIACRYPT 2017, Part I*, *LNCS* 10624, pages 502–532. Springer, Heidelberg, December 2017. (Page 7.)

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011*, *LNCS* 6597, pages 253–273. Springer, Heidelberg, March 2011. (Pages 33 and 34.)

[13] We point out that these are exactly the steps of $\mathcal{S}_{\text{Setup}}$.

BW07. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC 2007*, *LNCS* 4392, pages 535–554. Springer, Heidelberg, February 2007. (Pages 6, 7, 8, 28, and 31.)

Can01. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. (Page 39.)

Can03. R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. https://eprint.iacr.org/2003/239. (Pages 6, 31, 33, and 35.)

Can20. R. Canetti. Universally composable security. *J. ACM*, 67(5), September 2020. (Page 33.)

CKLM12a. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In *EUROCRYPT 2012*, *LNCS* 7237, pages 281–300. Springer, Heidelberg, April 2012. (Pages 10 and 29.)

CKLM12b. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. Cryptology ePrint Archive, Report 2012/012, 2012. https://eprint.iacr.org/2012/012. (Page 29.)

CL01. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT 2001*, *LNCS* 2045, pages 93–118. Springer, Heidelberg, May 2001. (Page 5.)

DGK$^+$21. I. Damgård, C. Ganesh, H. Khoshakhlagh, C. Orlandi, and L. Siniscalchi. Balancing privacy and accountability in blockchain identity management. In *CT-RSA 2021*, *LNCS* 12704, pages 552–576. Springer, Heidelberg, May 2021. (Page 5.)

DHO16. I. Damgård, H. Haagh, and C. Orlandi. Access control encryption: Enforcing information flow with cryptography. In *TCC 2016-B, Part II*, *LNCS* 9986, pages 547–576. Springer, Heidelberg, October / November 2016. (Page 7.)

DOT18. P. Datta, T. Okamoto, and K. Takashima. Adaptively simulation-secure attribute-hiding predicate encryption. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 640–672. Springer, Heidelberg, December 2018. (Pages 6, 8, 12, 28, and 39.)

EG14. A. Escala and J. Groth. Fine-tuning Groth-Sahai proofs. In *PKC 2014*, *LNCS* 8383, pages 630–649. Springer, Heidelberg, March 2014. (Page 28.)

FLA06. K. B. Frikken, J. Li, and M. J. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *NDSS 2006*. The Internet Society, February 2006. (Page 5.)

For87. L. Fortnow. The complexity of perfect zero-knowledge (extended abstract). In *19th ACM STOC*, pages 204–209. ACM Press, May 1987. (Page 9.)

GMR88. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988. (Page 8.)

GMW87. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987. (Page 9.)

GS08. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, *LNCS* 4965, pages 415–432. Springer, Heidelberg, April 2008. (Pages 28 and 29.)

JSI96. M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT'96*, *LNCS* 1070, pages 143–154. Springer, Heidelberg, May 1996. (Page 6.)

KLM$^+$18. S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu. Function-hiding inner product encryption is practical. In *SCN 18*, *LNCS* 11035, pages 544–562. Springer, Heidelberg, September 2018. (Pages 33, 34, and 39.)

KPW15. E. Kiltz, J. Pan, and H. Wee. Structure-preserving signatures from standard assumptions, revisited. In *CRYPTO 2015, Part II*, *LNCS* 9216, pages 275–295. Springer, Heidelberg, August 2015. (Pages 28 and 29.)

KSW08. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT 2008*, *LNCS* 4965, pages 146–162. Springer, Heidelberg, April 2008. (Pages 6, 7, 8, 11, 28, 29, and 31.)

LDB03. N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. In *22nd ACM PODC*, pages 182–189. ACM, July 2003. (Page 5.)

MM15. C. Matt and U. Maurer. A definitional framework for functional encryption. In *CSF 2015 Computer Security Foundations Symposium*, pages 217–231. IEEE Computer Society Press, 2015. (Pages 33 and 34.)

MPR11. H. K. Maji, M. Prabhakaran, and M. Rosulek. Attribute-based signatures. In *CT-RSA 2011*, *LNCS* 6558, pages 376–392. Springer, Heidelberg, February 2011. (Pages 3 and 6.)

Nak09. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. manuscript, 2009. http://www.bitcoin.org/bitcoin.pdf. (Page 4.)

OT12a. T. Okamoto and K. Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *EUROCRYPT 2012*, *LNCS* 7237, pages 591–608. Springer, Heidelberg, April 2012. (Pages 8, 11, 28, 29, 39, and 48.)

OT12b. T. Okamoto and K. Takashima. Fully secure unbounded inner-product and attribute-based encryption. In *ASIACRYPT 2012*, *LNCS* 7658, pages 349–366. Springer, Heidelberg, December 2012. (Pages 8 and 28.)

Wee17. H. Wee. Attribute-hiding predicate encryption in bilinear groups, revisited. In *TCC 2017, Part I*, *LNCS* 10677, pages 206–233. Springer, Heidelberg, November 2017. (Pages 8 and 12.)

A Further Details on Related Work

Further details on matchmaking encryption. As described in the main body of this paper, it seems that PCS are a special case of both ME and A-ME, however this is not true for several reasons. Here, we show how one could try to match the requirements and then why this has to fail in general.

PCS treat the sender and receiver as symmetric, i.e., only having one key-generation procedure to produce the private keys for attributes. Furthermore, the policy is public and hence, when fixing sender and receiver attributes, the additional power to specify arbitrary policies does not have to be exercised to its full generality: Given a public policy (that specifies the policy sender and receiver must jointly fulfill), one can set the policy during encryption to be trivially satisfied, and have the decryption key to encode the specific sub-policy that results on evaluating the public policy on the receivers own attributes which leaves a simple condition of the sender's attributes. Finally, to be publicly verifiable, the receiver can prove in zero-knowledge that it is able to decrypt the given message m with its key from the authority, which could serve as a public signature that does not reveal the attributes (and no privacy on the payload m is required since a PCS scheme is a signature scheme). The above sequence of arguments sounds compelling, but are unfortunately not sound:

While we explained the stronger unforgeability requirements in the main body, we focus here on the privacy requirements since they seem to be more related due to the condition on attribute-hiding. Intuitively, this property says that nothing about the honest sender attributes is leaked beyond what a malicious receiver can infer bye decrypting the ciphertext with its own attributes and the policies it demands from the sender.

However, since ME and A-ME are tailored to the use-case of replacing a handshake including a single message, the adversary in the security game only obtains a single value (the ciphertext) which is a function of the private sender key. Due to this reason, there are schemes that are too weak for our purpose: consider an (A-)ME scheme with the following tweak: each encryption key is extended by a random bitstring r. If the message to be encrypted equals r, the attributes of the sender are revealed. Furthermore, we also append r to every ciphertext this (A-)ME scheme produces. Therefore, if only one ciphertext is ever generated, the privacy guarantees of the underlying scheme are preserved unless one guesses r, which can be made negligible. However, such a scheme could not be used in our context as there is an adaptive attack that directly reveals the attributes.

B On the Relationship Between PCS and PE

Our PCS scheme is based on predicate-only predicate encryption schemes. This is actually not a coincidence since there is a closer connection which we outline for the IND-based case. A PCS scheme $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ for the class $\mathcal{F}_\lambda$ of predicates $F : \mathcal{X}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}$ implies a collection of predicate-only PE schemes $\mathsf{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, namely for each of the associated class of predicates $\mathcal{F}'_{\lambda,F}$ defined by $f_x \colon \mathcal{X}_\lambda \to \{0,1\}$; $y \mapsto F(x,y)$. We consider here only the IND-based notion for PE (and PCS). In words, the resulting scheme is a predicate-only PE scheme for the sub-policy describing "to whom a party, identified by the secret key for f_x, can send messages allowed by the general policy". We assume that $x \in \mathcal{X}$ can be efficiently recovered from the description of f_x. In the resulting PE scheme (w.r.t. parameter F), a secret key for function f_x is equated with the secret functional key for (sender) attributes x of the associated PCS scheme. The core idea of the PE scheme is to treat the public keys (w.r.t. receiver attributes) of the PCS scheme as ciphertexts. Since a public key hides the attributes of a party, we satisfy the hiding property of PE directly. Decryption of a ciphertext is implemented by having the holder of the secret key for attribute x doing a trial signature to check whether it is allowed to sign messages for the public key encoded in the ciphertext. By the property of PCS, this is only possible if $f_x(y) = F(x, y) = 1$ and otherwise does not reveal anything beyond this fact. For completeness, we state the concrete set of algorithms (where we keep F explicit for clarity).

$\mathsf{PE}_F.\mathsf{Setup}(1^\lambda)$: Execute $(\mathsf{mpk}_{\mathsf{PCS}}, \mathsf{msk}_{\mathsf{PCS}}) \leftarrow \mathsf{PCS.Setup}(1^\lambda, F)$ and return $\mathsf{msk} := (\mathsf{mpk}_{\mathsf{PCS}}, \mathsf{msk}_{\mathsf{PCS}})$.

$\mathsf{PE}_F.\mathsf{KeyGen}(\mathsf{msk}, f_x)$: Execute $(\mathsf{pk}_{\mathsf{PCS}}, \mathsf{sk}_{\mathsf{PCS}}) \leftarrow \mathsf{PCS.KeyGen}(\mathsf{msk}_{\mathsf{PCS}}, x)$. Then define and return $\mathsf{sk}_{f_x} \leftarrow (\mathsf{mpk}_{\mathsf{PCS}}, \mathsf{pk}_{\mathsf{PCS}}, \mathsf{sk}_{\mathsf{PCS}})$.

$\mathsf{PE}_F.\mathsf{Enc}(\mathsf{msk}, x)$: Execute $(\mathsf{pk}_{\mathsf{PCS}}, \mathsf{sk}_{\mathsf{PCS}}) \leftarrow \mathsf{PCS.KeyGen}(\mathsf{msk}_{\mathsf{PCS}}, x)$. Return $\mathsf{ct} := \mathsf{pk}_{\mathsf{PCS}}$. (Note that we have $x \in \mathcal{X}_\lambda$ here.)

$\mathsf{PE}_F.\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$: Parse the secret key as $(\mathsf{mpk}_{\mathsf{PCS}}, \mathsf{pk}_{\mathsf{PCS}}, \mathsf{sk}_{\mathsf{PCS}})$. Run $\sigma \leftarrow \mathsf{PCS.Sign}(\mathsf{ct}, \mathsf{sk}_{\mathsf{PCS}}, m)$ for a random message m. Return the verification bit $\mathsf{Verify}(\mathsf{mpk}_{\mathsf{PCS}}, \mathsf{pk}_{\mathsf{PCS}}, \mathsf{ct}, m, \sigma)$.

In the remainder of this section, we leave the security parameter implicit to simplify notation.

Lemma B.1. *Let $F \in \mathcal{F}$ be a policy and let $\mathcal{F}'_F$ be the derived predicate class defined above. If* PCS *is an attribute-hiding PCS scheme for $\mathcal{F}$ then the scheme* PE_F *is an attribute hiding predicate-only PE scheme for $\mathcal{F}'_F$.*

Proof. The resulting scheme is a correct PE scheme by correctness of the PCS scheme. We now show that any valid adversary $\mathcal{A}$ against the PE scheme above, i.e., for the random experiment $\mathbf{AH}_\beta^{\mathsf{PE}_F}$ can be turned into a valid adversary $\mathcal{A}'$ for PCS experiment $\mathbf{AH}_\beta^{\mathsf{PCS}}$ that achieves the same advantage. The adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ is specified as follows: $\mathcal{A}'_1$ outputs the policy F. $\mathcal{A}'_2$ is called on input $\mathsf{mpk}_{\mathsf{PCS}}$ and has access to oracles $\mathsf{QKeyGenLR}_\beta(x_{i,0}, x_{i,1})$, $\mathsf{QCor}(i)$, and $\mathsf{QSign}(i, \mathsf{pk}', m)$. $\mathcal{A}'_2$ internally runs $\mathcal{A}$ and emulates the two PE oracles towards $\mathcal{A}$ as follows:

– $\mathsf{QEncLR}_\beta(x_{i,0}, x_{i,1})$: $\mathcal{A}'_2$ makes an oracle call $\mathsf{QKeyGenLR}_\beta(x_{i,0}, x_{i,1})$ and obtains the corresponding public key pk_i, which it returns as the ciphertext. We say that index i is honest.
– $\mathsf{O}(f_x) := \mathsf{KeyGen}(\mathsf{msk}, f_x)$: $\mathcal{A}'_2$ extracts the attributes x from the description of f_x and makes an oracle call $\mathsf{QKeyGenLR}_\beta(x, x)$. Let this oracle call be the jth oracle call and denote the response by pk_j. Then $\mathcal{A}'_2$ calls $\mathsf{QCor}(j)$ to obtain sk_j. Finally return the PE key $(\mathsf{mpk}_{\mathsf{PCS}}, \mathsf{pk}_j, \mathsf{sk}_j)$. We say that index j is corrupted.

When the internal emulation of $\mathcal{A}$ returns a bit α', $\mathcal{A}'_2$ outputs α'.

We first observe that thanks to the oracles of provided by the experiment $\mathbf{AH}_\beta^{\mathsf{PCS}}$, $\mathcal{A}'_2$ is able to simulate all key-generation queries perfectly and outputs keys identically distributed to the ones obtained from the PE scheme. For encryption queries w.r.t. a pair $(x_{i,0}, x_{i,1}$ of attributes, $\mathcal{A}'_2$, by invoking its own LR oracle, returns the public key pk corresponding to attribute x_β, where β is the parameter of the experiment. By definition, this is exactly what the oracle $\mathsf{QEncLR}_\beta(x_{i,0}, x_{i,1})$ would return in the experiment $\mathbf{AH}_\beta^{\mathsf{PE}_F}$ for the above PE scheme. Thus, the distribution of the output of $\mathcal{A}'_2$ in experiment $\mathbf{AH}_\beta^{\mathsf{PCS}}$ is identical to the distribution of the output of $\mathcal{A}$ in experiment $\mathbf{AH}_\beta^{\mathsf{PE}_F}$. This establishes $\mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PCS},\mathcal{A}'} = \mathsf{Adv}^{\mathrm{AH}}_{\mathsf{PE}_F,\mathcal{A}}$. It remains to analyze the validity of $\mathcal{A}'$, where we can assume that $\mathcal{A}$ is valid. This means for $\mathcal{A}$ that for all queries $(x_{i,0}, x_{i,1})$ to the simulated oracle $\mathsf{QEncLR}_\beta(\cdot,\cdot)$ and for any function f queried to the simulated key generation oracle $\mathsf{KeyGen}(\mathsf{msk},\cdot)$, we have $f(x_{i,0}) = f(x_{i,1})$. By definition of the admissible predicates f, consider an f that led to the jth query $\mathsf{QKeyGenLR}_\beta(x_j, x_j)$ by $\mathcal{A}'_2$, where x_j is the attributed extracted from f. By construction and the definition of the predicate class, this index j is corrupted and the condition on $\mathcal{A}$ implies that for each such x_j we have

$$f_{x_j}(x_{i,0}) = F(x_j, x_{i,0}) = f_{x_j}(x_{i,1}) = F(x_j, x_{i,1}) \tag{2}$$

for all attribute pairs $(x_{i,0}, x_{i,1})$ corresponding to honest indexes.

In order for $\mathcal{A}'$ to be valid, two conditions must hold. Recall that $\mathcal{QC}$ is the set of corrupted indexes j and their keys, and $\mathcal{QK}$ is the set of all keys.

1. for every $(j, \mathsf{pk}_j, \mathsf{sk}_j, x_{j,0}, x_{j,1}) \in \mathcal{QC}$ and for all $(i, \mathsf{pk}_i, \mathsf{sk}_i, x_{i,0}, x_{i,1}) \in \mathcal{QK}$, it must hold that $\underbrace{x_{j,0} = x_{j,1} =: x_j}_{(*)}$ and $\underbrace{F(x_j, x_{i,0}) = F(x_j, x_{i,1})}_{(**)}$,
2. and for all $(i, \mathsf{pk}_i, \mathsf{pk}_j, m, \sigma) \in \mathcal{QS}$, and $(i, \mathsf{pk}_i, \mathsf{sk}_i, x_{i,0}, x_{i,1}), (j, \mathsf{pk}_j, \mathsf{sk}_j, x_{j,0}, x_{j,1}) \in \mathcal{QK}$, we have $F(x_{i,0}, x_{j,0}) = F(x_{i,1}, x_{j,1})$.

We observe that Item 2 is trivially satisfied since $\mathcal{A}'$ does not call its signature oracle. Condition $(*)$ holds since $\mathcal{A}'$ only corrupts indexes j for which it called $\mathsf{QKeyGenLR}_\beta(x_j, x_j)$ beforehand, and for which condition therefore $(**)$ trivially holds. For all remaining honest indexes, we observe that $(**)$ is satisfied because we already established Eq. (2). This concludes the proof. $\square$

The analogous statement holds in the simulation-based world:

Lemma B.2. *Let $F \in \mathcal{F}$ be a policy and let $\mathcal{F}'_F$ be the derived predicate class defined above. If PCS is a simulation-based attribute-hiding PCS scheme for $\mathcal{F}$ then the scheme PE_F is a simulation-based attribute hiding predicate-only PE scheme for $\mathcal{F}'_F$.*

Proof (Sketch). We are more brief here and only sketch how to obtain a simulator $\mathcal{S}^{\mathsf{PE}_F} = (\mathcal{S}^{\mathsf{PE}_F}_{\text{Setup}}, \mathcal{S}^{\mathsf{PE}_F}_{\text{Enc}}, \mathcal{S}^{\mathsf{PE}_F}_{\text{KG}})$ based on the assumed simulator $\mathcal{S}^{\mathsf{PCS}} = (\mathcal{S}^{\mathsf{PCS}}_{\text{Setup}}, \mathcal{S}^{\mathsf{PCS}}_{\text{KG}}, \mathcal{S}^{\mathsf{PCS}}_{\text{Cor}}, \mathcal{S}^{\mathsf{PCS}}_{\text{Sgn}})$.

– $\mathcal{S}^{\mathsf{PE}_F}_{\text{Setup}}$ runs $(\mathsf{mpk}, s_{\mathsf{PCS}}) \leftarrow \mathcal{S}^{\mathsf{PCS}}_{\text{Setup}}(F)$ and returns $(\mathsf{mpk}, s_{\mathsf{PCS}})$. The simulator(s) further maintain a leakage set $\mathcal{L}_{\mathsf{PCS}}$ which is initially empty (note that the experiment for PE will maintain the leakage set $\mathcal{L}_{\mathsf{PE}}$). Furthermore, the simulators maintain two (initially empty tables T_e and T_k and a counter C, initially 1.
– The simulator $\mathcal{S}^{\mathsf{PE}_F}_{\text{KG}}$ receives as input the state s, leakage set $\mathcal{L}_{\mathsf{PE}}$, and on the ith query to this oracle the requested function f (which is not hidden from the simulator). It first sets $T_k[i] := C$ and extracts the attributes x from the function description and stores $x_C := x$.
  The leakage set contains the following (new) information: for all previous encryption queries, $j \in [n_{ENC}]$ (where n_{ENC} is the number of PE encryption queries so far), the mappings $(j, i) \mapsto f(x_j) = F(x, x_j) =: b_j$ are found in $\mathcal{L}_{\mathsf{PE}}$ (where the last equality follows by definition of the predicate class).
  We now first add the $i-1$ mappings $(T_k[1], C) \mapsto F(x_{T_k[1]}, x_C), \ldots, (T_k[i-1], C) \mapsto F(x_{T[i-1]}, x_C)$ to $\mathcal{L}_{\mathsf{PCS}}$ (for all the previously stored attributes $x_{T_k[\cdot]}$ extracted from functions to the key-gen oracle). Then, $\mathcal{S}^{\mathsf{PCS}}_{\text{KG}}(s_{\mathsf{PCS}}, \mathcal{L}_{\mathsf{PCS}})$ is invoked, which returns a value pk and updates the state s_{PCS}. The simulator $\mathcal{S}^{\mathsf{PE}_F}_{\text{KG}}$ updates its state accordingly, increments the counter C and proceeds as follows:
  $\mathcal{S}^{\mathsf{PE}_F}_{\text{KG}}$ updates the leakage set $\mathcal{L}_{\mathsf{PCS}}$ by including the triple (C, x_C, M) where the set M (of new mappings) is formed as the union of the two sets $\{(C, T_e[j]) \mapsto b_j\}_{j \in [n_{ENC}]}$ and

$\{(C, T_k[j]) \mapsto F(x_C, x_{T_k[j]})\}_{j=1..i}$. (Note that this union covers all "receiver" indexes ever created in the view of $\mathcal{S}^{\mathsf{PCS}}$.) Then, $\mathcal{S}^{\mathsf{PCS}}_{\text{Cor}}(s_{\mathsf{PCS}}, \mathcal{L}_{\mathsf{PCS}}, C)$ is invoked, which returns a value sk and updates the state s_{PCS}. The simulator $\mathcal{S}^{\mathsf{PE}_F}_{\text{KG}}$ updates its state accordingly, and outputs $(\mathsf{mpk}, \mathsf{pk}, \mathsf{sk})$ as the functional secret key for f.

– The simulator $\mathcal{S}^{\mathsf{PE}_F}_{\text{Enc}}$ receives as input the state s and leakage set $\mathcal{L}_{\mathsf{PE}}$. On the ith query to the oracle (w.r.t. the unknown (to the simulator) attributes x), the leakage set contains the following (new) information: for all previous key-generation queries, $j \in [n_{KG}]$ (where n_{KG} is the number of PE key-generation queries so far), the mappings $(i, j) \mapsto f_j(x) = F(x_j, x) =: b_j$ are found in $\mathcal{L}_{\mathsf{PE}}$ (where the last equality follows by definition of the predicate class). Thus, all the new mappings $(T_k[j], C) \mapsto b_j$ are added to $\mathcal{L}_{\mathsf{PCS}}$ and then $\mathcal{S}^{\mathsf{PCS}}_{\text{KG}}(s_{\mathsf{PCS}}, \mathcal{L}_{\mathsf{PCS}})$ is invoked, which returns the value pk and updates the state s_{PCS}. The simulator $\mathcal{S}^{\mathsf{PE}_F}_{\text{Enc}}$ updates its state accordingly, sets $T_e[i] := C$, increments the counter C, and finally returns pk.

As we can observe, the modular structure of the PE scheme from the PCS scheme is reflected in the design of the simulator. Based on our intuition on the PE scheme, it is not surprising that the leakage towards the PCS simulator upon a PE Encryption query is exactly the policy evaluations of this "newly created party" i (in the role of the receiver) with respect to all previously "corrupted parties" (in the role of the sender). In the view of the PCS simulator, all indexes j are considered corrupted, for which a PE key-generation query has been submitted. Furthermore, on a PE key-generation query, we create a new party with the attributes x extracted from the function that is received as part of this query (recall that the PE notion we look at here is not function hiding). Thus, also this leakage can be simulated since it is easy to figure out to whom this new party can send (and hence to include all the relevant mappings), since it either receives the PE leakage set (that refer to predicate evaluations w.r.t. PE encryptions) and for all other created parties, their attributes are known (as the only other case is that the party index was created as part of a PE key-gen query where we could extract the attributes). To make this mapping precise, we have to keep tables T_e and T_k which map each respective query to the PE encryption oracle resp. PE key-generation oracle to its party-index in the view of the PCS experiment (since PE key-gen is implemented similarly to the proof of the previous lemma by creating a party using PCS key-gen and subsequent corruption). It is thus not hard to see that if we have a distinguisher $\mathcal{A}$ with advantage α in distinguishing $\mathbf{Real}^{\mathsf{PE}_F}(1^\lambda, \mathcal{A})$ and $\mathbf{Ideal}^{\mathsf{PE}_F}(1^\lambda, \mathcal{A}, \mathcal{S}^{\mathsf{PE}})$ for the simulator above, this implies a distinguisher $\mathcal{A}'$ with the same advantage for the systems $\mathbf{Real}^{\mathsf{PCS}}(1^\lambda, \mathcal{A})$ and $\mathbf{Ideal}^{\mathsf{PCS}}(1^\lambda, \mathcal{A}, \mathcal{S}^{\mathsf{PCS}})$. Similar to the previous lemma, $\mathcal{A}'$ only has to emulate encryption and key-generation queries towards $\mathcal{A}$. The former are implemented using PCS key-generation queries, and the latter are implemented using key-generation queries followed by a corruption query. □

C From Single-Challenge to Multi-Challenge Attribute Hiding Predicate Encryption

We need a predicate encryption scheme that allows an adversary to adaptively query for many challenge ciphertexts via a left-or-right oracle, see Definition 2.9. We below present a secret-key version the definition from [OT12a] that allows for a single challenge query and gives access to an encryption oracle and a key generation oracle. Afterwards, we prove, via a hybrid argument, that this implies the notion we use in this work.

Definition C.1 (Single-Challenge Indistinguishability-Based Attribute Hiding). *Let* $\mathsf{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PE scheme for a function family* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$*. For* $\beta \in \{0, 1\}$*, we define the experiment* $\text{SC-AH}^{\mathsf{PE}}_\beta$ *in Fig. 12, where the advantage of an adversary*

```
$\textbf{SC-AH}_\beta^{\mathsf{PE}}(1^\lambda, \mathcal{A})$
$\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$
$(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathsf{Enc}(\mathsf{msk},\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot)}(1^\lambda)$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, x_\beta)$
$\alpha \leftarrow \mathcal{A}_2^{\mathsf{Enc}(\mathsf{msk},\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{st}, \mathsf{ct})$
$\textbf{Output: } \alpha$
```

Fig. 12: Single-challenge attribute-hiding game of PE.

$\mathcal{A}$ is defined as

$$\mathsf{Adv}_{\mathsf{PE},\mathcal{A}}^{\text{SC-AH}}(\lambda) = \left|\Pr[\text{SC-AH}_0^{\mathsf{PE}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\text{SC-AH}_1^{\mathsf{PE}}(1^\lambda, \mathcal{A}) = 1]\right|.$$

We call an adversary valid *if for the output pair (x_0, x_1) and for any function f queried to the key generation oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$, we have $f(x_0) = f(x_1)$ (with probability 1 over the randomness of the adversary and the involved algorithms).*

A predicate-only predicate encryption scheme PE is called single-challenge attribute hiding *if for any valid polynomial-time adversary $\mathcal{A}$, there exists a negligible function* negl *such that $\mathsf{Adv}_{\mathsf{PE},\mathcal{A}}^{\text{SC-AH}}(\lambda) \leq \text{negl}(\lambda)$.*

Theorem C.2 (From Single-Challenge to Multi-Challenge Attribute Hiding Predicate Encryption). *Let $\mathsf{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a PE scheme for a function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$. If PE is single-challenge attribute hiding, then it is also attribute hiding. More precisely, for any PPT adversary $\mathcal{A}$, there exits a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}_{\mathsf{PE},\mathcal{A}}^{\text{AH}}(\lambda) = q \cdot \mathsf{Adv}_{\mathsf{PE},\mathcal{B}}^{\text{SC-AH}}(\lambda),$$

where q is an upper bound on the number of challenge queries to QEncLR_β by $\mathcal{A}$.

Proof. For every $i \in \{0, \dots, q\}$, we define the game G_i that corresponds to $\text{AH}_\beta^{\mathsf{PE}}$, where the oracle QEncLR_β is replaced by an oracle that for the first i queries corresponds to QEncLR_0, and afterwards corresponds to QEncLR_1. Note that $\mathsf{G}_0 = \text{AH}_1^{\mathsf{PE}}$ and $\mathsf{G}_q = \text{AH}_0^{\mathsf{PE}}$.

For $i \in \{1, \dots, q\}$, we define an adversary $\mathcal{B}_i$ for the $\text{SC-AH}^{\mathsf{PE}}$ game as follows: Emulate an execution of $\mathcal{A}$ and forward all KeyGen queries to the KeyGen oracle of $\mathcal{B}_i$. For the first $i - 1$ queries (x_0, x_1) adversary $\mathcal{A}$ makes to QEncLR_β, forward x_0 to the oracle Enc. When $\mathcal{A}$ makes the ith such query, return (x_0, x_1) to the $\text{SC-AH}^{\mathsf{PE}}$ challenger and give the returned challenge ciphertext to $\mathcal{A}$. For all remaining queries to QEncLR_β, $\mathcal{B}_i$ forwards x_1 to the oracle Enc. Finally, $\mathcal{B}_i$ outputs the same α as $\mathcal{A}$.

Note that if $\mathcal{B}_i$ is playing the game $\text{SC-AH}_0^{\mathsf{PE}}$, it emulates the game G_i to $\mathcal{A}$, and if it is playing the game $\text{SC-AH}_1^{\mathsf{PE}}$, it emulates the game G_{i-1} to $\mathcal{A}$. Hence,

$$\Pr[\text{SC-AH}_\beta^{\mathsf{PE}}(1^\lambda, \mathcal{B}_i) = 1] = \Pr[\mathsf{G}_{i-\beta}(1^\lambda, \mathcal{A}) = 1].$$

Further note that $\mathcal{B}_i$ is a valid $\text{SC-AH}^{\mathsf{PE}}$ adversary if $\mathcal{A}$ is a valid AH^{PE} adversary: The output pair (x_0, x_1) corresponds to a QEncLR_β query from $\mathcal{A}$. Validity of $\mathcal{A}$ thus implies that for any function f queried to the key generation oracle KeyGen, we have $f(x_0) = f(x_1)$, which is the validity condition for $\mathcal{B}_i$.

We finally define the adversary $\mathcal{B}$ to pick $i \in [q]$ uniformly at random an then execute $\mathcal{B}_i$. Since all $\mathcal{B}_i$ are valid SC-AH$^{\mathsf{PE}}$ adversaries, so is $\mathcal{B}$. Furthermore,

$$\begin{aligned}
\mathsf{Adv}^{\text{SC-AH}}_{\mathsf{PE},\mathcal{B}}(\lambda) &= \left|\Pr\left[\text{SC-AH}^{\mathsf{PE}}_0(1^\lambda,\mathcal{B})=1\right] - \Pr\left[\text{SC-AH}^{\mathsf{PE}}_1(1^\lambda,\mathcal{B})=1\right]\right| \\
&= \left|\sum_{i=1}^{q} \Pr[i=q]\cdot\Pr\left[\text{SC-AH}^{\mathsf{PE}}_0(1^\lambda,\mathcal{B}_i)=1\right] - \sum_{i=1}^{q} \Pr[i=q]\cdot\Pr\left[\text{SC-AH}^{\mathsf{PE}}_1(1^\lambda,\mathcal{B}_i)=1\right]\right| \\
&= \frac{1}{q}\cdot\left|\sum_{i=1}^{q}\Pr\left[\mathsf{G}_i(1^\lambda,\mathcal{A})=1\right] - \sum_{i=1}^{q}\Pr\left[\mathsf{G}_{i-1}(1^\lambda,\mathcal{A})=1\right]\right| \\
&= \frac{1}{q}\cdot\left|\Pr\left[\mathsf{G}_q(1^\lambda,\mathcal{A})=1\right] - \Pr\left[\mathsf{G}_0(1^\lambda,\mathcal{A})=1\right]\right| \\
&= \frac{\mathsf{Adv}^{\text{AH}}_{\mathsf{PE},\mathcal{A}}(\lambda)}{q}.
\end{aligned}$$

This implies $\mathsf{Adv}^{\text{AH}}_{\mathsf{PE},\mathcal{A}}(\lambda) = q\cdot\mathsf{Adv}^{\text{SC-AH}}_{\mathsf{PE},\mathcal{B}}(\lambda)$ and concludes the proof. □