Updatable Policy-Compliant Signatures

Christian Badertscher [1], Monosij Maitra [*2],
Christian Matt [†4], and Hendrik Waldner [‡5]

[1]Input Output, Switzerland – christian.badertscher@iohk.io
[2]Indian Institute of Technology Kharagpur, India – monosij@cse.iitkgp.ac.in
[4]Primev, Switzerland – christian@primev.xyz
[5]University of Maryland, College Park, USA– hwaldner@umd.edu

Abstract

Policy-compliant signatures (PCS) are a recently introduced primitive by Badertscher et al. [TCC 2021] in which a central authority distributes secret and public keys associated with sets of attributes (e.g., nationality, affiliation with a specific department, or age) to its users. The authority also enforces a policy determining which senders can sign messages for which receivers based on a *joint* check of their attributes. For example, senders and receivers must have the same nationality, or only senders that are at least 18 years old can send to members of the computer science department. PCS further requires attribute-privacy – nothing about the users' attributes is revealed from their public keys and signatures apart from whether the attributes satisfy the policy or not. The policy in a PCS scheme is fixed once and for all during the setup. Therefore, a policy update requires a redistribution of all keys. This severely limits the practicality of PCS. In this work, we introduce the notion of *updatable policy-compliant signatures (UPCS)* extending PCS with a mechanism to efficiently update the policy without redistributing keys to all participants.

We define the notion of UPCS and provide the corresponding security definitions. We then provide a generic construction of UPCS based on digital signatures, a NIZK proof system, and a so-called secret-key two-input partially-hiding predicate encryption (2-PHPE) scheme. Unfortunately, the only known way to build the latter for general two-input predicates is using indistinguishability obfuscation. We show that the reliance on the heavy tool of 2-PHPE is inherent to build UPCS by proving that non-interactive UPCS implies 2-PHPE.

To circumvent the reliance on 2-PHPE, we consider *interactive* UPCS, which allows the sender and receiver to interact during the message signing procedure. In this setting, we present two schemes: the first one requires only a digital signature scheme, a NIZK proof system, and secure two-party computation. This scheme works for arbitrary policies, but requires sender and receiver to engage in a two-party computation protocol for each policy update. Our second scheme additionally requires a (single-input) predicate-encryption scheme but, in turn, only requires a single interaction between sender and receiver, independent of the updates. In contrast to 2-PHPE, single-input predicate encryption for certain predicate classes is known to exist (e.g., from pairings) under more concrete and well-understood assumptions.

[*]Work done at Ruhr University Bochum and MPI-SP, Germany, and also partially at TU Darmstadt, Germany.
[†]Work done partially while author was at Concordium, Switzerland.
[‡]Work done partially while author was at the Max Planck Institute for Security and Privacy, Germany.

Contents

1 Introduction

Policy-compliant signatures (PCS) [BMW21] are credential-based, enhanced signature schemes in which a policy F governs signature generation: a party with attributes x (encoded in a privacy-preserving way in their public-key) that wishes to sign a message m destined for a receiving party with attributes y, is only ever able to produce a valid signature certifying this action if and only if $F(x, y) = 1$. This notion extends attribute-based signatures [MPR11] and policy-based signatures [BF14] (which allow for policies based solely on the sender's attributes) and has interesting applications in corporate environments and financial applications including payment systems [BMW21, BSW23]. This is due to the fact that this type of signature scheme is able to merge the act of signing (to unlock funds) with cryptographic compliance checks while still being publicly verifiable. Basic examples include: enforcing that both parties are of a minimal legal age and operate in jurisdictions between which no sanctions exist, or to ensure that certain tokens are only spent in a prescribed context, such that between employees (e.g. of a state) and subsidized facilities/services where they obtain discounts. The scheme ensures that nothing more than the mere validity of the statement leaks, keeping the receiver's attributes hidden from the sender/signer at all times, meeting the promise to enforce compliance to a certain rule set in a privacy-preserving way. The policy in such systems can either be defined by a certain legal system (and implemented by an accredited credential issuer), or purely application-driven by a provider to govern the spending rights of domain-specific tokens. On a technical level, in a PCS scheme, a (credential) authority generates a master public and secret-key pair for a given policy, and then derives public and secret keys for each user and their attributes from the master secret key. A major practical limitation is that the policy is fixed at setup time (when generating the master keys) and, consequently, all users need to receive new key pairs if the policy is changed, incurring a high communication cost. In the above mentioned applications, regularly new rules are put in place, which demands PCS to be more dynamic. Therefore, we introduce *updatable policy-compliant signatures (UPCS)* which allow authorities to update policies more efficiently, i.e., without updating the users' keys.

1.1 Updatable PCS Description

Like a regular PCS scheme, a UPCS scheme consists of a Setup algorithm that an authority can use to generate a master public and secret key for an initial policy F_0. For a set of attributes x, the authority can then use the master secret key to generate a key pair corresponding to these attributes using KeyGen. An updatable PCS scheme has an additional procedure PolUpd that takes the master secret key and a new policy F' as an input and outputs a so-called update token $\mathsf{tok}_{F'}$. Using the current update token, the secret key of the sender, and the public key of the receiver, the sender can sign a message by executing Sign. Everyone can then verify the signature using the master public key, the current update token and the public keys of the sender and receiver with the Verify procedure. The signature is valid if the signature and message have not been modified, and if the attributes of the sender and receiver satisfy the current policy associated with the update token $\mathsf{tok}_{F'}$.

Security requirements. As for regular policy-compliant signatures, UPCS security consists of two parts: unforgeability and privacy. Unforgeability captures what one can expect from any signature scheme, i.e., that no valid signature for a new message can be produced without knowledge of the secret key, even given access to a signing oracle for arbitrary messages. Additionally, unforgeability for (updatable) policy-compliant signatures prevents an adversary from creating a signature that verifies for a pair of public keys for a sender and a receiver with attributes not satisfying the policy.

For *updatable* PCS, we require these properties to hold even if the adversary can adaptively update the policies. Furthermore, we require that an adversary is not able to generate a valid update token on its own. In particular, an adversary is prevented from producing a valid signature for an update token of an old policy, or an update token generated on its own, and attributes that do not satisfy it, even if these attributes satisfy a newer policy.

Privacy in this context refers to hiding the attributes of sender and receiver. This means an adversary, seeing signatures for a sender and receiver for different policies, should not learn anything about their attributes except for which of the considered policies they satisfy and what can be deduced from that. This must still hold if one of the two parties is corrupted, i.e., the attributes of the receiver are protected from a malicious sender and vice versa. We refer to this privacy property as attribute-hiding.

Below, we motivate and provide some contextualization on policy updatability and then dive into the challenges and overview of our contributions.

1.2 On Policy Updatability

We start by discussing the importance of updates in the context of PCS [BMW21] for its practical usage in more detail. UPCS improves the applicability of PCS and covers a blind spot in some novel applications of digital signatures. Badertscher et al. [BMW21] laid the foundations for PCS and discussed such applications in detail, and Badertscher et al. [BSW23] provide the first estimates on its practicality. Our motivation for UPCS is to present it in an enriched model (e.g., with interactivity) which is necessary to unleash its potential in many more applications. We believe that policy-updatability is a critical stepping stone to push PCS further towards practice.

From the UPCS description above, we note that it is indeed necessary that the signer and the verifier need to be aware of the current update token. It is not the case though that all the parties in the system have to do some work with respect to the update token and there is no key update that needs to be executed. The only work that is needed in this step is done by the authority, which is the generation of the update token. The complexity of the signing and the verification procedure remains the same for any update token. To minimize communication cost and make every user in the system aware of the current update token, the authority can post this token, for example, on a public bulletin-board. Or, when using a blockchain, which is one area where (U)PCS could be applied to secure transactions [BMW21], the most recent update token can be simply published inside the current block. This ensures that all parties in the system are aware of the same token. Further, our UPCS model does not enforce any specific requirements on the policy-update procedure, it is a standard probabilistic algorithm that can, in principle, produce the same update token multiple times. Hence, our definitions (and constructions) also allow for the reusability of update tokens.

We also note that UPCS can capture certain basic forms of revocation mechanisms by design – an updated policy may embed checks for revoked sender/receiver's attributes that never satisfy the new policy jointly with that of other parties in the system. In contrast, verifier-local revocation [ISE+18, LLNW14, BS04] requires verifiers to possess updated revocation lists. As UPCS signatures are valid, subject to policies being satisfied *jointly* by sender and receiver's attributes, such revocation lists can vary across different verifiers and thus could get cumbersome to maintain. Similarly, server-aided revocation [CM21, LNWZ19, CDLQ16, NWZ16] is also inefficient as it additionally requires an (untrusted) server, whereas the authority is supposed to update revocation lists periodically.

Finally, we do not consider delegation for UPCS in this paper. That is, we require all the updates to be executed by the authority. Nevertheless, one could imagine that the authority desires to delegate certain updates, i.e., allowing another party to generate update tokens for a *subclass* of the functionality. We leave investigating such delegation capabilities as future work.

1.3 Challenges and Interactivity

Non-interactive constructions. Policy-compliant signatures, as introduced in [BMW21], are non-interactive in the sense that after receiving all relevant keys from the authority, parties can locally sign messages for arbitrary receivers and the resulting signatures can be verified by all other parties given only the pre-distributed public keys. In this work, we provide an updatable PCS scheme fitting this narrative. Our non-interactive construction relies on a regular signature scheme, a non-interactive zero-knowledge proof system, and a *secret-key two-input partially hiding predicate encryption (2-PHPE)* scheme. The latter is introduced in this work and corresponds to a weakened variant of two-input PE. The class of policies supported by our UPCS scheme directly corresponds to the set of predicates that the 2-PHPE scheme supports (see the technical overview below for more details).

Our 2-PHPE needs to satisfy *strong* attribute-hiding (also known as *two-sided* security) [KSW08, FFMV23], where no information about the hidden attributes is revealed beyond the output of the predicate.[1] Unfortunately, such 2-PHPE schemes can currently only be obtained making the strong assumption of indistinguishability obfuscation (iO) [GGH+13, JLS21] which results in the immediate impracticality of the resulting schemes. This raises the question whether one can construct UPCS schemes without heavy tools such as 2-PHPE. We answer this question in the negative by proving that any non-interactive UPCS scheme can be used to construct a 2-PHPE scheme. Practically efficient non-interactive UPCS therefore appears to be out of reach with currently available techniques.

Interactive UPCS. Due to the negative result above, we relax the notion of UPCS to allow for some limited interaction among the parties. Considering blockchain transactions as the main motivation for PCS [BMW21], some sort of interaction between sender and receiver is already happening: before the sender can sign a transaction for a receiver, it has to obtain the correct address of the receiver and needs to agree on the amount to be transferred. The relevant information for an interactive UPCS scheme can thus be integrated into an extension of an existing payment process.

Based on this insight, we present two interactive UPCS schemes with a different tradeoff between interactivity and efficiency: the first scheme is very simple and is based only on regular signature schemes, a non-interactive zero-knowledge proof system and a two-party computation protocol. Furthermore, it supports arbitrary (efficiently computable) policies. The downside of this scheme is that it requires a single interaction between each sender and receiver every time the policy is updated. The second interactive scheme we present only requires a single interaction between sender and receiver and allows them to subsequently sign an unbounded number of messages even after policy updates. This scheme, however, relies on a predicate-encryption (PE) scheme, and the supported class of policies depends on the predicates supported by the PE scheme. Both schemes are presented in more detail in the technical overview below.

1.4 Technical Overview

1.4.1 Defining UPCS and its Security

Since we are interested in both, non-interactive as well as interactive UPCS schemes, we define UPCS in a generic way such that both settings are covered. Our definition models UPCS as

[1]Strong attribute-hiding (predicate-only) PE allows an adversary to obtain secret keys for predicates that *can* decrypt a challenge ciphertext, but it is still required to hide any other information about the attributes.

a message-driven protocol among a set of parties and a trusted authority $\mathfrak{A}$, all connected in a complete network consisting of pairwise point-to-point channels. Parties can receive instructions from the environment and subsequently can use the network to communicate with each other, and produce an output for the environment. We do not consider a dedicated adversary but consider the adversary to be part of the environment, as the dummy adversary in UC [Can01]. The environment can therefore corrupt any party, except for the trusted authority $\mathfrak{A}$, to obtain full control over it and learn all its secrets. For a fixed security parameter, we consider a family of attributes X, where $\mathcal{X}$ denotes the powerset of X, and a set of supported policies $\mathcal{F} = \{F \colon \mathcal{X} \times \mathcal{X} \to \{0,1\}\}$. In the beginning, the authority expects an input $(\textsc{Setup}, F_0)$ (from the environment) for some initial policy $F_0 \in \mathcal{F}$. It then produces a master key pair and an initial token tok_0. On input $(\textsc{Update}, F)$, the authority produces an update token tok_F. Furthermore, on input $(\textsc{KeyGen}, P_i, x_i)$, the authority produces a key pair for the attributes x_i for party P_i (using the master secret key). This formally spawns the new party P_i initialized with this new key pair. Any party P_S can then receive inputs $(\textsc{Sign}, \mathsf{tok}_F, \mathsf{pk}_R, m)$ instructing P_S to sign the message m for a receiver P_R with public key pk_R relative to the policy update token tok_F, producing a signature σ. Any party (not just P_R) can then verify the signature via $(\textsc{Verify}, \mathsf{tok}_F, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$. Correctness of the UPCS scheme requires the result of the $\textsc{Verify}$ instruction to be equal to $F(x_S, x_R)$, where x_S and x_R are the attributes of the sender and receiver, respectively. That is, the signature is valid if and only if the attributes of the sender and receiver satisfy the policy F. A non-interactive UPCS scheme simply corresponds to a scheme in which signing and verifying signatures are local operations, whereas interactive UPCS schemes allow interaction between sender and receiver.

Security of a UPCS scheme covers *unforgeability* and *attribute-hiding*. Unforgeability means that any environment can trigger any of the following events only with negligible probability: $(\textsc{Verify}, \mathsf{tok}_F, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ returns 1 given pk_S or pk_R or tok_F that have not been output by $\textsc{KeyGen}$ or $\textsc{Update}$, pk_S and pk_R correspond to attributes not satisfying the policy F corresponding to tok_F, or the sender S is uncorrupted and has not been instructed to sign the message m.

To define the attribute-hiding property, we introduce an additional instruction $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ that internally does the same as $(\textsc{KeyGen}, P_i, x_{i,b})$ for a uniformly random bit b. The goal of the environment is then to guess the bit b and we call the scheme attribute-hiding if this is only possible with negligible advantage over random guessing. For this definition, we exclude queries that would trivially allow an adversary to distinguish. That is, a party P_i can only be corrupted if the tested attributes $x_{i,0}$ and $x_{i,1}$ are equal, and we require all policies to be equal for all attributes involved in a $\textsc{Sign}$ query, or belonging to corrupted senders (who could sign themselves).

1.4.2 Two-input Partially Hiding Predicate Encryption (2-PHPE)

We use secret-key 2-PHPE in Section 4.1 to construct a non-interactive UPCS scheme. For a set of attributes X with powerset $\mathcal{X}$, consider a class of predicates $\mathcal{P} = \{P \colon \mathcal{X} \times \mathcal{X} \to \{0,1\}\}$. A 2-PHPE scheme for $\mathcal{P}$ allows an authority to generate a master secret key msk using the algorithm Setup. The obtained msk can then be used to generate keys sk_P for predicates $P \in \mathcal{P}$ via $\mathsf{KeyGen}(\mathsf{msk}, P)$. The master secret key is also used to encrypt attributes $X \in \mathcal{X}$ and obtain a ciphertext $\mathsf{ct}_X = \mathsf{Encrypt}(\mathsf{msk}, X)$. In addition to encrypting, we also consider a method Encode to "encode" attributes $\widehat{X}$ using msk. In contrast to a ciphertext, an encoding $\mathsf{e}_{\widehat{X}}$ does not hide the attributes $\widehat{X}$ (hence the scheme is only partially hiding). Finally, there is an algorithm $\mathsf{Decrypt}$ that takes a secret key sk_P for a predicate P, an encoding e_{X_1} of X_1, and a ciphertext ct_{X_2} encrypting X_2 and outputs $P(X_1, X_2)$ with overwhelming probability.

Security of 2-PHPE is defined via a distinguishing game in which an adversary has access to $\mathsf{KeyGen}, \mathsf{Encode}$ and $\mathsf{Encrypt}$ oracles, subject to some admissibility conditions. In particular, the adversary can obtain encryptions of $X_{2,b}$, when submitting $X_{2,0}$ and $X_{2,1}$ for uniformly chosen $b \in \{0,1\}$, along with encodings e_{X_1} and secret keys sk_P satisfying $P(X_1, X_{2,0}) = P(X_1, X_{2,1})$. The scheme is called *partially strong attribute-hiding*[2] if no PPT adversary has non-negligible advantage of determining b over random guessing.

To the best of our knowledge, the only known way to instantiate such a 2-PHPE scheme for all predicates is via iO [GGH+13, JLS21]. Furthermore, a *sub-exponentially* secure 2-PHPE scheme supporting all boolean circuits actually implies iO. Even for a limited class of predicates (e.g., inner-products), a secret-key 2-PHPE scheme with strong attribute-hiding yields a non-interactive UPCS scheme for a class of policies captured by inner-products. One way to build such a 2-PHPE scheme is using trilinear maps, where the linearity can be used to evaluate the inner-product of the vectors provided by the authority, the sender, and the receiver. Weakening the assumption to build it using bilinear maps, while satisfying attribute-hiding, seems challenging – the evaluation between the attributes of these three parties seems to require trilinearity. We leave overcoming this issue as an interesting open problem and refer to Section 4.1 (particularly to Remarks 2 and 3) for more details on this.

1.4.3 Non-interactive UPCS from 2-PHPE

Given a 2-PHPE scheme for a class of predicates $\mathcal{P}$, we can construct a non-interactive UPCS scheme for the same class of policies $\mathcal{F} = \mathcal{P}$. Our construction additionally assumes a NIZK proof system and a digital signature scheme. A token for a policy $F \in \mathcal{F}$ consists of a 2-PHPE secret key for the predicate F, signed by the authority using the digital signature scheme. A public key for attributes X consists of a digital signature public key and a ciphertext of X, both signed by the authority. The corresponding secret key instead contains an encoding of X (since it does not have to hide X) and additionally the secret key of the digital signature scheme. When S wants to sign a message for a receiver R, S proves, using the NIZK, knowledge of a secret key with a valid signature from the authority containing an encoding e_{X_1} such that 2-PHPE decryption of e_{X_1} and the ciphertext ct_{X_2} in the public key of R returns 1.

Unforgeability of the scheme follows from the correctness of the 2-PHPE scheme, the soundness of the NIZK proof, and unforgeability of the digital signature scheme. The attribute-hiding property of the UPCS scheme follows from the security of the 2-PHPE scheme and the zero-knowledge property of the NIZK. Note that all 2-PHPE keys, ciphertexts, and encodings are generated by the authority. Thus, a secret-key 2-PHPE scheme is sufficient. Furthermore, it is only necessary to hide the attributes in the receiver's public key is sufficient since the sender's attributes are encoded in the UPCS secret key only known to the sender.

1.4.4 Non-interactive UPCS implies 2-PHPE

Since 2-PHPE is a heavy tool, it would be desirable to construct a UPCS scheme without assuming 2-PHPE. Unfortunately, this is impossible for non-interactive UPCS schemes. We prove this by constructing a 2-PHPE scheme based on a non-interactive UPCS scheme, where the set of supported predicates exactly matches the set of supported policies of the assumed UPCS scheme. The basic idea of the construction is as follows: a 2-PHPE secret key for a policy P corresponds to a UPCS update token for the policy corresponding to P. To encode a set of attributes X_1, we generate a UPCS key pair for that attribute set, and the encoding consists of both the secret and public keys.

[2]Note that the adversary may even learn sk_P for which $P(X_1, X_{2,b}) = 1, \forall b \in \{0,1\}$. This is referred to as strong attribute-hiding for the 2-PHPE scheme. For brevity, we call this attribute-hiding in the rest of the paper (except Section 4.1, where we again clarify this formally).

To encrypt X_2, we also generate a UPCS key pair, but only include the public key in the ciphertext, to ensure that X_2 remains hidden. To decrypt a pair $(\mathsf{e}_{X_1}, \mathsf{ct}_{X_2})$, we use the secret key in e_{X_1} and the public key in ct_{X_2} to generate a UPCS signature (on a fixed message). We then verify this signature and output the bit that $\mathsf{UPCS.Verify}$ returns. Correctness of the UPCS scheme implies that the generated signature is valid if and only if $P(X_1, X_2) = 1$, which implies correctness of the 2-PHPE scheme. Security of the 2-PHPE scheme also directly follows from the attribute-hiding property of the UPCS scheme.

1.4.5 UPCS with Interaction after each Policy Update

If we allow the sender and receiver to interact when signing for the first time after a policy update, it is rather easy to obtain a UPCS scheme. The basic idea is to let the parties jointly generate a NIZK proof that their attributes satisfy the current policy. More concretely, the scheme assumes a basic digital signature scheme, a NIZK proof system, and a secure two-party computation (2PC) protocol and works as follows: on Setup, the authority generates a CRS for the NIZK and key pairs of the digital signature scheme. The master public key consists of the CRS value and all signature public keys, the master secret key contains the corresponding secret keys, and the initial update token is a signature on the initial policy F_0 under the authority's signing key. Update simply consists of signing the new policy. In the first step of KeyGen, for attributes x, a regular signature key pair is generated. The final generated public key consists of the signature verification key signed by the authority, and the secret key consists of the signing key and the signed pair of verification key and attributes x. When a sender P_S wants to sign a message for a receiver P_R for the first time under the current policy, the two execute the 2PC protocol to produce a NIZK proof π for the statement that P_S and P_R know valid signatures from the authority on their verification keys together with attributes that satisfy the current policy. The UPCS signature of a message m then consists of π and a regular signature under the signing key of P_S of m, π, and the verification key of P_R. For future signatures, under the same policy, the proof π can be reused. Verification of a UPCS signature entails verifying the standard signature and the NIZK proof.

Unforgeability of the UPCS scheme is implied by unforgeability of the basic signature scheme and soundness of the NIZK proof. The zero-knowledge property of the proof further implies that a UPCS signature does not reveal anything about the attributes of the sender and receiver beyond the fact that they satisfy the current policy. Security of the 2PC finally implies that the attributes of the sender are hidden from the receiver and vice versa.

1.4.6 UPCS with One-Time Interaction

In the last step, we construct a UPCS scheme that requires only a single interaction between a pair of sender and receiver such that the sender can afterwards sign arbitrarily many messages for that receiver, even under future policy updates. This scheme combines ideas from our non-interactive scheme and the interactive one, but only uses a standard single-input predicate-encryption (PE) scheme. As in our non-interactive scheme, a policy update token consists of a signed PE secret key. Since UPCS policies take the attributes of the sender and receiver as separate inputs and the PE scheme only takes a single input, we concatenate the attributes of the sender and receiver into a single element and generate a PE secret key for the corresponding predicate that splits the input in half and then evaluates the corresponding two-input policy. In more detail, for a two-input policy F' that takes as an input two attribute sets X_S and X_R, we create a single-input function $\widehat{F}'$ that takes as an input a single set $X_{S,R}$, which is the disjoint union of the sender and receiver attributes, i.e. $X_{S,R} \coloneqq \{(x, 0) | x \in X_S\} \cup \{(x, 1) | x \in X_R\}$. The function $\widehat{F}'$ then splits the input set $X_{S,R}$ into the two sets X_S and X_R, where attributes in X_S are of the form $(x, 0)$ and attributes in X_R are of the form $(x, 1)$, and then executes $F'(X_S, X_R)$. The set of supported policies in this scheme corresponds to all policies F' for which the PE scheme supports the corresponding $\widehat{F'}$. A UPCS secret key for a set of attributes X contains a signature from the authority on X together with a secret key for a digital signature scheme, and the corresponding public key contains the signed corresponding digital signature public key.

When a sender S wants to sign a message for a receiver R for the first time, they interact in a 2PC to first compute a PE ciphertext for the pair of attribute sets (X_S, X_R). Afterwards, they produce a NIZK proof that the ciphertext was generated correctly using attributes with valid signatures from the authority for the corresponding public keys of S and R. To sign a message, the sender then produces a NIZK proof of knowledge of such a ciphertext and that the ciphertext decrypts to 1 using the PE secret key in the current policy update token.

If the scheme is implemented directly as described above, then, for the final signing of the message, the sender is required to generate a NIZK proof over another NIZK proof, namely that the known ciphertext comes with a valid NIZK proof from the 2PC that it is an encryption of the correct attributes. Since this can be rather inefficient (depending on the used NIZK scheme), we slightly modify the scheme to avoid this recursive proof. A seemingly simple solution would be to add the ciphertext generated in the 2PC in plain to a signature and then only prove that this ciphertext decrypts to 1. The problem with this approach is that now everyone can evaluate this ciphertext for all policies for which an update token exists, and not just those for which the sender generates a signature. This scheme therefore does not satisfy our privacy notion that only allows information leakage about the attributes that trivially follow from the usage of the scheme, and, in particular, hide policy evaluations for policies under which no signature has been generated. We avoid this issue by generating a commitment to the ciphertext in the 2PC, together with a proof that the commitment was generated correctly, and then add this commitment to each signature. A signature then contains two NIZK proofs, where the first one is taken directly from the 2PC, and the second one proves that the commitment is to a (secret) ciphertext that decrypts to 1 using the PE secret key of the current policy.

Instantiating PE. We also require a (strong) attribute-hiding, adaptively secure PE scheme to instantiate the UPCS scheme with one-time interaction presented above. Such PE schemes exist based on pairings for inner-product (in the standard model) [OT12] and quadratic predicates (in the generic group model) [BCFG17, RPB$^+$19]. We refer the reader to Section 5.2.1 for more details about the instantiation. The fact that the underlying PE schemes can be based on pairings highlights the practical potential of our UPCS scheme with one-time interaction, i.e., one can, in principle, implement and benchmark the presented UPCS scheme, given efficient digital signatures and NIZK proofs that are compatible with the underlying pairing-based PE schemes. We leave a practical instantiation of this scheme as future work.

2 Preliminaries

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 $a \coloneqq A(x; r)$ 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 $\mathcal{B}(q)$. For any probabilistic event E, we denote its complement by $\overline{\mathsf{E}}$. Furthermore, for the last construction that we present in the main body, we introduce the following notation:

$$X_s \dot\cup X_r \coloneqq \{(x, 0) | x \in X_s\} \cup \{(x, 1) | x \in X_r\}.$$

2.1 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 Figure 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 $\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda)$ for a negligible function $\text{negl}(\cdot)$.

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

Figure 1: Existentially Unforgeability Game of DS.

2.2 Non-Interactive Zero Knowledge Proofs

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

Definition 2.3 (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 2.4 (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:

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

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

Definition 2.5 (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, $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ a pair of algorithms (the simulator), with $\mathsf{Sim}'(\mathsf{CRS}, \tau, x, w) = \mathsf{Sim}_2(\mathsf{CRS}, \tau, x)$ for $(x, w) \in R$, and $\mathsf{Sim}'(\mathsf{CRS}, \tau, x, w) = \mathsf{failure}$ for $(x, w) \notin R$. For $\beta \in \{0, 1\}$, we define the experiment $\text{ZK}^{\mathsf{NIZK}}_\beta(1^\lambda, \mathcal{A})$ in Figure 2. The associated advantage of an adversary $\mathcal{A}$ is defined as

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

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

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

Figure 2: Zero-knowledge property of NIZK.

Besides the notion of zero-knowledge and soundness, we also introduce the notion of extractability as in [CKLM12].

Definition 2.6 (Extractability)**.** Let $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a NIZK proof system for a relation R and the corresponding language L, $E = (E_1, E_2)$ be a pair of algorithms (the extractor) which shares a secret state. 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

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

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).

2.3 Predicate Encryption

The notion of *predicate-only predicate encryption* has first been introduced by Katz et al. [KSW08]. At a high level, a predicate encryption scheme allows, given the master secret key, to generate secret keys sk_f for functions f in the supported function family $\mathcal{F}$. Using the master public 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 then yields a constant, say 1, if $f(x) = 1$, and another constant, say 0, if $f(x) \neq 1$. 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 which results in an additional negligible error.

Definition 2.7 (Predicate-Only Predicate Encryption)**.** Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of sets $\mathcal{F}_\lambda$ of predicates $f : \mathcal{X}_\lambda \to \{0, 1\}$. 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 a master public-secret key pair $(\mathsf{mpk}, \mathsf{msk})$.

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

$\mathsf{Enc}(\mathsf{mpk}, x)$**:** Takes as input the master public key mpk 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 $(\mathsf{mpk}, \mathsf{msk})$ 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

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

As a security requirement, we want the ciphertexts to hide the encrypted attributes, for which we define an indistinguishability-based notion below.

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

Figure 3: Attribute-Hiding game of PE.

2.3.1 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 [OT12]. While the definition in [OT12] allows the adversary to obtain only a single challenge ciphertext, we use a generalization of this notion that allows for multiple challenges. A simple/standard hybrid argument for public-key functional encryption can be used to show this implication.

Definition 2.8 (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 Figure 3, 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 $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_\beta)$.

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

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

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 $\mathsf{Adv}_{\mathsf{PE},\mathcal{A}}^{\mathrm{AH}}(\lambda) \leq \mathrm{negl}(\lambda)$.

2.4 Commitment Schemes

We recap the definition of a commitment scheme as stated in [Lin10] together with the definition of computational hiding, perfect binding and equivocality.

Definition 2.9 (Commitment Scheme)**.** A commitment scheme consists of a setup algorithm Setup that takes as an input a unary representation of the security parameter and outputs a CRS CRS and a PPT algorithm Com that takes as an input a CRS CRS, a message m, some randomness r and outputs a commitment com.

We call the pair (m, r) the decommitment of com.

We recall that commitments are secure, w.r.t. the security definitions below, under parallel composition, which we use in the proof of our predicate encryption based interactive UPCS scheme.

Definition 2.10 (Hiding of CS)**.** Let $(\mathsf{Setup}, \mathsf{Com})$ be a commitment scheme, then we define the experiment $\mathrm{HIDE}_\beta^{\mathsf{Com}}$ in Figure 4. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined as:

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

A commitment scheme Com is called computationally hiding, if for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ it holds that $\mathsf{Adv}_{\mathsf{Com},\mathcal{A}}^{\mathrm{HIDE}}(\lambda) \leq \mathrm{negl}(\lambda)$.

```
$\mathbf{HIDE}^{\mathsf{Com}}_{\beta}(1^\lambda, \mathcal{A})$
$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$
$(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{CRS})$
$r \leftarrow \{0,1\}^\lambda$
$c = \mathsf{Com}(\mathsf{CRS}, m_\beta; r)$
$\alpha \leftarrow \mathcal{A}_2(\mathsf{st}, c)$
**Output:** $\alpha$
```

Figure 4: Hiding Game for a commitment scheme.

```
$\mathbf{BIND}^{\mathsf{Com}}(1^\lambda, \mathcal{A})$
$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$
$(c, m, r, m', r') \leftarrow \mathcal{A}(\mathsf{CRS})$
**Output:** 1 if $\mathsf{Com}(\mathsf{CRS}, m; r) =$
$\mathsf{Com}(\mathsf{CRS}, m'; r')$
        and 0 otherwise
```

Figure 5: Binding Game for a commitment scheme.

Definition 2.11 (Binding of CS)**.** Let $(\mathsf{Setup}, \mathsf{Com})$ be a commitment scheme, then we define the experiment $\mathrm{BIND}^{\mathsf{Com}}$ in Figure 5. A commitment scheme Com is called perfectly binding, if for any adversary $\mathcal{A}$ it holds that $\Pr[\mathrm{BIND}^{\mathsf{Com}}(1^\lambda, \mathcal{A}) = 1] = 0$.

Definition 2.12 (Equivocality)**.** Let $\mathsf{com} = (\mathsf{Setup}, \mathsf{Com})$ be a commitment scheme and $\mathsf{Eq} = (\mathsf{Eq}_1, \mathsf{Eq}_2, \mathsf{Eq}_3)$ a triple of algorithms (the equivocator), then we define:

$\mathsf{QCom}(m)$**:** On input m, sample $r \leftarrow \{0,1\}^\lambda$, compute $c := \mathsf{Com}(\mathsf{CRS}, m; r)$ and output (c, r).

and set $\mathsf{Eq}'(m) = (c := \mathsf{Eq}_2(\mathsf{CRS}), r := \mathsf{Eq}_3(c, m))$. For $\beta \in \{0,1\}$, we define the experiment $\mathrm{Eq}^{\mathsf{com}}_{\beta}(1^\lambda, \mathcal{A})$ in Figure 6. The associated advantage of an adversary $\mathcal{A}$ is defined as

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

A commitment scheme com is called equivocal, with respect to an equivocator $\mathsf{Eq} = (\mathsf{Eq}_1, \mathsf{Eq}_2, \mathsf{Eq}_3)$, if $\mathsf{Adv}^{\mathrm{Eq}}_{\mathsf{com},\mathcal{A},\mathsf{Eq}}(\lambda) \leq \mathrm{negl}(\lambda)$ for all PPT algorithms $\mathcal{A}$.

$\mathbf{Eq}^{\mathsf{com}}_0(1^\lambda, \mathcal{A}, \mathsf{Eq})$	$\mathbf{Eq}^{\mathsf{com}}_1(1^\lambda, \mathcal{A}, \mathsf{Eq})$
$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$	$\mathsf{CRS} \leftarrow \mathsf{Eq}_1(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QCom}(\cdot)}(\mathsf{CRS})$	$\alpha \leftarrow \mathcal{A}^{\mathsf{Eq}'(\cdot)}(\mathsf{CRS})$
Output: α	**Output:** α

Figure 6: Equivocality of com.

We note that this is different to the standard definition of equivocality. We require the existence of an oracle since, in the proof of our construction, we need to equivocate multiple commitments at once. We observe that we can realize this potentially stronger notion by simply committing to the message m using a commitment scheme with randomness r in the commit phase and output a zero-knowledge proof that the message m with randomness r leads to the commitment, as a decommitment. The equivocator in this scheme simply runs the simulator of the NIZK proof system.[3]

[3]We note that the decommitment for this schemes consists of more then just the randomness r used to generate this commitment. For simplicity, in the remainder of this work, we simply denote the decommitment as the randomness r.

2.5 Two-Party Computation

In this section, we introduce the preliminaries on UC-secure MPC [GS18]. A secure two-party computation protocol is carried out between two parties P_1 and P_2 (modeled as interactive Turing machines) and is associated with a function specified as a circuit C. Party P_1 has input x_1 and P_2 has input x_2. At the end of the protocol, P_1 gets the output. If the function is randomized, the parties contribute randomness as part of their input. We follow the real/ideal world paradigm to formalize security of a two-party computation protocol Π_{2PC} secure against malicious adversaries. First, we describe the ideal process.

2.5.1 Ideal Process

The ideal world is associated with a trusted party and parties P_1, P_2. At most one of P_1, P_2 is controlled by an adversary. The process proceeds in the following steps:

Inputs: Each party obtains its input, party P_1 obtains x_1 and party P_2 obtains x_2 from the environment.

Inputs to Trusted Party: The parties send their inputs to the trusted party $\mathcal{F}_C$. The honest party sends the same input it received from the environment to the trusted party. The adversary, however, can send a different input to the trusted party.

Aborting Adversaries: An adversarial party can send a message to the trusted party to abort the execution. Upon receiving this, the trusted party terminates the ideal world execution.

Trusted party produces output to adversary: Suppose the trusted party receives inputs x_1' and x_2' from P_1 and P_2 respectively, then it evaluates $\mathsf{out} = C(x_1', x_2')$. If P_1 is corrupted, output out to the adversary. Otherwise, activate the adversary with the empty string.

Output to parties: If the party P_1 is honest and the execution has not been aborted, $\mathcal{F}_C$ outputs out to P_1.

We denote the adversary participating in the above protocol to be Sim. We define $\mathsf{Ideal}^{\Pi_{\mathsf{2PC}}}_{\mathcal{Z},C,\mathsf{Sim}}$ to be the distribution of the output of the environment executing the ideal process.

2.5.2 Real Process

In the real process both the parties execute the protocol Π_{2PC}. At most one of P_1 or P_2 is controlled by an adversary. We denote the adversarial party to be $\mathcal{A}$. As in the ideal process, they receive inputs from the environment. We define $\mathsf{Real}^{\Pi_{\mathsf{2PC}}}_{\mathcal{Z},C,(P_1,P_2)}$ to be the joint distribution over the outputs of the adversary and the honest party.

We define the security of two-party computation as follows:

Definition 2.13. Consider a two-party computation specified as a circuit C. Let Π_{2PC} be a two-party protocol for evaluating C. We say that Π_{2PC} *securely computes* C if for every malicious PPT adversary $\mathcal{A}$ in the real world, there exists a PPT adversary Sim (called the simulator) in the ideal world such that for any enviornment $\mathcal{Z}$,

$$\mathsf{Ideal}^{\Pi_{\mathsf{2PC}}}_{\mathcal{Z},C,\mathsf{Sim}} \approx_c \mathsf{Real}^{\Pi_{\mathsf{2PC}}}_{\mathcal{Z},C,\mathcal{A}}$$

3 A Model for Interactive and Non-Interactive PCS with Updates

Here, we sketch a simple extension to the PCS setting that includes the possibility to have interactive processes between protocol participants.

3.1 Model Basics

General protocol execution. We consider a standard real-world protocol execution as in [Can01] with a dynamic set of parties that are able to pass messages to each other. A protocol is an interactive program that specifies how incoming messages and inputs are processed, what outputs are generated and which messages are sent to which party. Protocols can be triggered (and protocol machines spawned) at a party by an external input, and/or due to incoming messages from the network. Upon each of these events, the protocol specification prescribes how information is processed, how the local state of the party is changed, whether output is given to the caller, or whether new messages are sent onto the network. Protocols can further trigger the start of sub-protocols, and several of these sub-protocols can be run simultaneously by any single party.

We follow the dynamic execution model defined in [Can01] where we simply merge the environment and the adversary into one entity (which is without loss of generality due to the completeness of the dummy adversary in UC). We thus assume a PPT adversary $\mathcal{A}$ that has full control over the basic communication network in that it can read, delay, delete, modify, and infect messages on the basic network. For some actions, parties are assumed to be connected by *secure channels*, in which case the adversary only learns that a message is being sent over that link without leaking the contents, and without the ability of the attacker to modify the content. The adversary also controls the scheduling of protocol events, including the initiation of protocols and providing their external inputs. The adversary can at any time decide to corrupt a party by a special request $\textsc{Corrupt}$ (except the trusted third party), in which case the entire private state (unless explicitly erased) is revealed together with all incoming and outgoing messages by this party. From this point on, the adversary is in full control of the protocol machine.

A general model for PCS with updates. We first define some notation. Let $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of attributes 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 $\mathcal{F}_\lambda = \{F\colon \mathcal{X}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}\}$ of policies/predicates. We omit λ in the subscripts from now on for the sake of simplicity and writing.

Updatable policy-compliant signatures can be modeled as message-driven protocols, denoted UPCS, where the set of parties consists of an authority $\mathfrak{A}$ and a set of users $P_1, \ldots, P_n$. The authority is a trusted entity that generates the setup of the system and generates the key-pair for the parties based on the attributes they are assigned to. More precisely, the authority is initially triggered by an input $(\textsc{Setup}, F)$ upon which it generates and outputs a master public key mpk and an additional token tok_0, an auxiliary value in anticipation of future policy updates. If the setup completes without failure, we say that the authority is initialized with policy F. After setup, the authority accepts inputs of the form $(\textsc{KeyGen}, P_i, x_i)$ upon which it has to generate the public-private key pair together with party P_i, potentially running an interactive sub-protocol over private channels with P_i. Such a user that completed this process, and thereby obtains a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ together with its attributes and the master public key, is called initialized (with attributes x_i). Formally the user P_i signals the successful initialization by outputting $(\textsc{Initialized}, \mathsf{pk}_i)$.

For policy updates, the authority accepts inputs of the form $(\textsc{Update}, F')$ and must output a so-called update token tok_e, where e is an increasing index corresponding to the number of times a policy has been updated.

Once a user is initialized, it accepts two types of inputs in order to sign and verify signatures. On input $(\textsc{Sign}, \mathsf{tok}, R, m)$ to party S, the party generates a signature σ, potentially after exchanging messages (i.e., running a sub-protocol) with the receiver R. Finally, on input $(\textsc{Verify}, \mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ for any combination of token, keys, message, and signature, the party locally produces a decision bit, where 0 indicates that the signature is invalid, and 1 indicates that the signature is valid with respect to the master public key mpk.

Concrete protocol descriptions in this work. Not all steps above involve an interactive process between parties and hence many of the inputs to the protocol machines are answered directly by running algorithms which we can define following and extending the definition of [BMW21]. In fact, our interactive schemes leverage a 2PC during signing between sender S and receiver R, while setup, key-generation, and verification are local operations (where in case of key-generation the keys are transmitted to the enrolled user). Thus, in order to define a full UPCS protocol, it is sufficient to specify the following tuple of algorithms:

$\mathsf{Setup}(1^\lambda, F)$**:** On input a unary representation of the security parameter λ and an initial policy $F \in \mathcal{F}_\lambda$, output a master public and secret key pair $(\mathsf{mpk}, \mathsf{msk})$, and an initial token tok. This algorithm is run by $\mathfrak{A}$ upon input $(\textsc{Setup}, F)$.

$\mathsf{PolUpd}(\mathsf{msk}, \mathsf{mpk}, F')$**:** On input the master secret and public key, msk and mpk, and a new policy F', output an update token tok. This algorithm is run by $\mathfrak{A}$ upon input $(\textsc{Update}, F')$.

$\mathsf{KeyGen}(\mathsf{msk}, P_i, x_i)$**:** On input the master secret key msk and a set of attributes $x_i \in \mathcal{X}_\lambda$ for party P_i, output a public and secret key pair $(\mathsf{pk}, \mathsf{sk})$. This algorithm is run by $\mathfrak{A}$ upon input $(\textsc{KeyGen}, P_i, x_i)$. The authority then sends $(\mathsf{mpk}, \mathsf{pk}, \mathsf{sk})$ to party P_i over a secure channel.

$\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$**:** This is a deterministic algorithm, that on input a master public key mpk, a token tok, two public keys $\mathsf{pk}_S, \mathsf{pk}_R$, a signature σ on a message m, output either 0 or 1. This algorithm is executed upon input $(\textsc{Verify}, \mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ by party P_i.

Finally, in the general case of an interactive signing process, we will define the following subprotocol:

$\Pi_{\text{sign}}(\mathsf{mpk}, \mathsf{tok}, \mathsf{sk}_S, \mathsf{pk}_R, R, m)$**:** On input a master public key mpk, a token tok, a sender secret key sk_S and public key pk_R, a receiver identity, and a message m, this subprotocol, after termination, produces an output for the sender that is either a signature σ or $\perp$ (and no output for R). This subprotocol is invoked by S upon input $(\textsc{Sign}, \mathsf{tok}, R, m)$.

In case signing is a non-interactive process as in [BMW21], then the subprotocol Π_{sign} boils down to specifying an algorithm Sign just as in [BMW21], where we assume that any public key pk_R issued by the authority $\mathfrak{A}$ towards a party P_i is associated to its identity in a certifiable way.[4] Obtaining a public key for a given identity can be accomplished by any out-of-band communication mechanism.

$\mathsf{Sign}(\mathsf{mpk}, \mathsf{tok}, \mathsf{sk}_S, \mathsf{pk}_R, m)$**:** On input a master public key mpk, a token tok, a sender secret key sk_S, a receiver public key pk_R, and a message m, output either a signature σ or $\perp$. This is run by S upon input $(\textsc{Sign}, \mathsf{tok}, R, m)$, and where pk_R is the key issued to R by the authority (e.g. obtained from R).

[4]For example via a signature on the pair (pk_{P_i}, P_i).

3.2 Correctness and Security for PCS with Updates

PCS Correctness with Policy Updates. We first define the correctness of a (potentially interactive) updatable PCS scheme.

Definition 3.1. An updatable UPCS protocol π is called correct if the following property holds with overwhelming probability in the above experiment with respect to any PPT adversary: if d is the output by some initialized and honest party P_i upon input $(\textsc{Verify}, \mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ and if it holds that

1. tok is the value output by $\mathfrak{A}$ on input $(\textsc{Update}, F)$ or $(\textsc{Setup}, F)$ for some policy F.
2. the public keys pk_S and pk_R belong to initialized honest parties P_S and P_R computed as a result of invocations $(\textsc{KeyGen}, P_S, x_S)$, $(\textsc{KeyGen}, P_R, x_R)$, respectively; and
3. the signature σ is the returned value by party P_S on input $(\textsc{Sign}, \mathsf{tok}, R, m)$,

then it must hold that $d = F(x_S, x_R)$.

Unforgeability. We capture unforgeability in this setting following [BMW21], but cast it in our execution model.

Definition 3.2. An updatable UPCS protocol π is called unforgeable if no PPT adversary $\mathcal{A}$ is able to provoke the following event E_{forge} with better than negligible probability in the above experiment. Event E_{forge} occurs if an honest party P_i outputs a decision bit $d = 1$ on input $(\textsc{Verify}, \mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ where the following property holds:

1. The public key pk_S is associated with some initialized party P_S that is not corrupted and was never invoked on input $(\textsc{Sign}, \mathsf{tok}, R, m)$; or
2. the update token tok has never been computed by the authority A on any input $(\textsc{Update}, F)$ or $(\textsc{Setup}, F)$; or
3. the public key pk_S (resp. pk_R) does not belong to any initialized party S (resp. R) (by means of an invocation $(\textsc{KeyGen}, S, x_S)$, (resp. $(\textsc{KeyGen}, R, x_R)$) to $\mathfrak{A}$); or
4. there is a policy F such that tok is the output of $\mathfrak{A}$ on input $(\textsc{Update}, F)$ or $(\textsc{Setup}, F)$, and there are attributes x_S and x_R associated to some initialized parties S and R with public keys pk_S and pk_R, respectively (as the result of inputs $(\textsc{KeyGen}, S, x_S)$ and $(\textsc{KeyGen}, R, x_R)$ to $\mathfrak{A}$), such that $F(x_S, x_R) = 0$.

Attribute-Hiding. To define the attribute-hiding experiment, we extend the usual capabilities of the adversary by allowing it to perform a *test-key-gen* query. That is, in addition to its regular actions, the adversary $\mathcal{A}$ can initialize a party alternatively by $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$. At the onset of the experiment, a bit b is chosen uniformly at random. Upon a test-query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$, party P_i executes the protocol action as if it received input $(\textsc{KeyGen}, P_i, x_{i,b})$. In the following, we treat inputs of the form $(\textsc{KeyGen}, P_i, x_i)$ made by $\mathcal{A}$ as if the adversary asked $(\textsc{Test-KeyGen}, P_i, (x_i, x_i))$. The adversary is allowed to execute all regular actions as defined above, but is not allowed to perform certain actions, involving any party P_i to which a test query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ has been issued, that would violate any of the conditions below:

1. $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ is only issued for a corrupted P_i if $x_{i,0} = x_{i,1}$.
2. Vice-versa, the party P_i can only be corrupted in the execution if
   (a) it holds that $x_{i,0} = x_{i,1}$; and
   (b) For all other initialized parties P_j, initialized via $(\textsc{Test-KeyGen}, P_j, (x_{j,0}, x_{j,1}))$, it must hold that $F(x_{i,0}, x_{j,0}) = F(x_{i,1}, x_{j,1})$ for all policies F for which there is either the input $(\textsc{Setup}, F)$ or an input $(\textsc{Update}, F)$ to $\mathfrak{A}$.
3. For any signature produced by P_i via input $(\textsc{Sign}, \mathsf{tok}, P_j, m)$, where tok is an update token for policy F output by $\mathfrak{A}$ on input $(\textsc{Update}, F)$ or $(\textsc{Setup}, F)$, it must hold that $F(x_{i,0}, x_{j,0}) = F(x_{i,1}, x_{j,1})$.

An adversary is called valid, if it obeys the conditions with probability 1 over the randomness of the adversary and the protocol machines. At the end of its run, $\mathcal{A}$ outputs a bit b' (as a guess for b).

Definition 3.3. An updatable UPCS protocol π is called attribute-hiding if, for any valid PPT adversary $\mathcal{A}$ in the above extended experiment, the probability of event $b = b'$ is at most $1/2 + \text{negl}(\lambda)$.

4 Non-Interactive Updatable Policy-Compliant Signatures

4.1 Two-Input Partially Hiding (Predicate-Only) Predicate Encryption

In this section, we formally define the notion of a two-input partially hiding (predicate-only) predicate encryption (2-PHPE) scheme that we use to build our non-interactive UPCS scheme in Section 4.2. As the name suggests, our 2-PHPE definition considers attribute-hiding only with respect to one slot of the predicate. We discuss this further in Remark 2 after formally defining the primitive.

Definition 4.1 (Two-input Partially-Hiding Predicate-Only Predicate Encryption)**.** Let $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of attributes and let $\mathcal{X}_\lambda$ denote the powerset of X_λ. Further, let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of two-input predicate sets $\mathcal{P}_\lambda = \{P\colon \mathcal{X}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}\}$. A *two-input partially-hiding predicate-only predicate encryption* (2-PHPE) scheme for the predicate class $\mathcal{P}_\lambda$ is given by a tuple of five PPT algorithms 2-PHPE = (2-PHPE.Setup, 2-PHPE.KeyGen, 2-PHPE.Encode, 2-PHPE.Encrypt, 2-PHPE.Decrypt):

$\mathsf{Setup}(1^\lambda)$**:** On input a unary representation of the security parameter λ , output public parameters pp[5] and a master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, P)$**:** On input msk and a predicate $P \in \mathcal{P}_\lambda$, output a secret key sk_P.

$\mathsf{Encode}(\mathsf{msk}, x_1)$**:** On input msk and an attribute set $x_1 \in \mathcal{X}_\lambda$ (for slot 1), output an *encoding* e_{x_1} of x_1. (Note that e_{x_1} may not hide x_1.)

$\mathsf{Encrypt}(\mathsf{msk}, x_2)$**:** On input msk and an attribute set $x_2 \in \mathcal{X}_\lambda$ (for slot 2), output a ciphertext ct.

$\mathsf{Decrypt}(\mathsf{sk}_P, (\mathsf{e}_{x_1}, \mathsf{ct}))$**:** On input a secret key sk_P, an encoded value e_{x_1} for slot 1 and a ciphertext ct for slot 2, output 0, 1 or $\perp$.

[5]We assume pp to be an implicit input in all the other algorithms.

Correctness. A 2-PHPE scheme for $\mathcal{P}_\lambda$ is correct, if for all $(x_1, x_2) \in (\mathcal{X}_\lambda)^2, P \in \mathcal{P}_\lambda$, for all pp and msk in the support of $\mathsf{Setup}(1^\lambda)$, all secret keys sk_P in the support of $\mathsf{KeyGen}(\mathsf{msk}, P)$, it holds that $\Pr[\mathsf{Decrypt}(\mathsf{sk}_P, (\mathsf{Encode}(\mathsf{msk}, x_1), \mathsf{Encrypt}(\mathsf{msk}, x_2))) = P(x_1, x_2)] \geq 1 - \mathrm{negl}(\lambda)$, where the probability is over the random coins of Encode and Encrypt.

Definition 4.2 (Indistinguishability-Based Partial Attribute-Hiding)**.** Let 2-PHPE = (Setup, KeyGen, Encode, Encrypt, Decrypt) be a 2-PHPE scheme for a predicate class $\mathcal{P}_\lambda$ as defined in Definition 4.1. For $\beta \in \{0, 1\}$ and for an adversary $\mathcal{A}$, we define an experiment $\mathrm{AH}_\beta^{\text{2-PHPE}}(1^\lambda, \mathcal{A})$ as shown below.

```
$\mathbf{AH}_\beta^{\text{2-PHPE}}(1^\lambda, \mathcal{A})$
$(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{Encode}(\mathsf{msk},\cdot),\mathsf{QEncLR}_\beta(\cdot,\cdot)}(1^\lambda, \mathsf{pp})$
**Output:** $\alpha$
```

Figure 7: Partially-Hiding game of 2-PHPE.

The oracles $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ and $\mathsf{Encode}(\mathsf{msk}, \cdot)$ work exactly as described in Definition 4.1 above. For any attribute-set pair $(x_2^0, x_2^1) \in \mathcal{X}^2$ for slot 2, define the oracle $\mathsf{QEncLR}_\beta(x_2^0, x_2^1) :=$ $\mathsf{Encrypt}\left(\mathsf{msk}, x_2^\beta\right)$. The adversary $\mathcal{A}$ in the above experiment may make an arbitrary polynomial number of queries *adaptively* to all its oracles before it outputs α. We call the adversary $\mathcal{A}$ *valid*, if for all predicates $P \in \mathcal{P}_\lambda$ queried to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$, for all attribute sets $x_1 \in \mathcal{X}$ queried to $\mathsf{Encode}(\mathsf{msk}, \cdot)$ and for all attribute-pairs $(x_2^0, x_2^1) \in \mathcal{X}^2$ queried to $\mathsf{QEncLR}_\beta(\cdot, \cdot)$, it holds that $P(x_1, x_2^0) = P(x_1, x_2^1)$. We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathrm{AH}}_{\text{2-PHPE},\mathcal{A}}(1^\lambda) = |\Pr[\mathrm{AH}_0^{\text{2-PHPE}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathrm{AH}_1^{\text{2-PHPE}}(1^\lambda, \mathcal{A}) = 1]|.$$

A 2-PHPE scheme satisfies *partial attribute-hiding* security, if for any valid PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that $\mathsf{Adv}^{\mathrm{AH}}_{\text{2-PHPE},\mathcal{A}}(1^\lambda) \leq \mathrm{negl}(\lambda)$.

Remark 1 (Asymmetric Evaluation). From Definition 4.1, we note that it is important that the evaluation of any predicate P is *asymmetric*. That is, given sk_P, an encoding e_{x_1} of attributes x_1 and a ciphertext ct encrypting attributes x_2, it is possible to learn $P(x_1, x_2)$ only and *not* $P(x_2, x_1)$. This is generally ensured by the syntax of any multi-input FE or PE scheme by explicitly fixing the index associated to a ciphertext w.r.t. a slot [AJ15, BKS16, AYY22, FFMV23]. For example, one way to do this for an n-input FE scheme is to have the encryption algorithm take the index $i \in [n]$ as an input along with the message and generate the slot i ciphertext as ct_i [BKS16]. We keep this implicit in Definition 4.1 by denoting the attribute set for slot i as x_i for $i \in [2]$.

Remark 2 (Partial Attribute-Hiding). Note that for the first slot, the "encoding" e_{x_1} for attributes x_1 may not hide x_1, but still allows the evaluation of any predicate P, given a secret key sk_P and a ciphertext ct encrypting attributes x_2 with respect to the second slot. However, this encoding procedure requires the 2-PHPE master secret key and is thus a secret operation. The 2-PHPE security definition above further allows the adversary $\mathcal{A}$ to learn secret keys that decrypt to 1. That is, for any of $\mathcal{A}$'s encoding query x_1 (to the $\mathsf{Encode}(\mathsf{msk}, \cdot)$ oracle) and any challenge query (x_2^0, x_2^1) (to the $\mathsf{QEncLR}_\beta(\cdot, \cdot)$ oracle), $\mathcal{A}$ may possess secret keys sk_P such that $P(x_1, x_2^b) = 1$ for all $b \in \{0, 1\}$. Definition 4.2 thus requires (strong) attribute-hiding, but only with respect to the second slot.

Remark 3 (2-PHPE Instantiations and Implications). As stated in Remark 1, our 2-PHPE requires (strong) attribute-hiding for slot 2 whereas there are no privacy requirements on slot 1. 2-PHPE is thus a weaker variant of (and is implied by) the general notion of two-input predicate encryption (2-PE) – a primitive that has recently been defined and studied under the more general framework of multi-input attribute-based and predicate encryption [AYY22, FFMV23, ARYY23, ATY23]. However, we believe there is no two-input (strong) attribute-hiding PE even for any specific class of predicates in any model under any assumption.[6] (Realizing even 2-PHPE with strong attribute-hiding for inner-product predicates seems to require trilinear maps, which is a strong assumption. At a high level, a trilinear map can allow the computation of the inner-product functionality between vectors from three parties – the first and second party along with the secret key holder.) Currently, the only known way to build a (strong) attribute-hiding 2-PE (and 2-PHPE) scheme for all circuits is via iO for circuits [GGH$^+$13, JLS21] through two-input FE [GGG$^+$14, AJ15, BV15].

In terms of implications, we note that sub-exponentially secure, (strong) attribute-hiding 2-PHPE (even in the secret-key setting) for all circuits actually implies iO. To see this, we observe that (strong) attribute-hiding 2-PHPE implies PE with the same security (by simply ignoring the first input). Furthermore, (strong) attribute-hiding PE implies FE (where a secret key for some function with, say, n output bits corresponds to n PE secret keys for the individual bits). Finally, sub-exponentially secure, collusion-resistant, secret-key FE for all circuits implies iO [KNT18].

4.2 Non-interactive UPCS Scheme

In this section, we present our first, non-interactive updatable policy-compliant signature scheme. This scheme relies on the two-input partially-hiding predicate encryption scheme introduced in the previous section. In this scheme, a public key of a party consists of a ciphertext ct encrypting its attributes as well as a verification key vk of a signature scheme. The corresponding secret key of this party contains the encoding of its attributes e_x and the signing key sk that corresponds to the verification key vk in the public key. Furthermore, the public key and the secret key of a party contain a signature of the authority that binds them together using the verification key vk, i.e., a signature in the public key that is generated for $(\mathsf{vk}, \mathsf{ct})$ and a signature in the secret key that is generated for $(\mathsf{vk}, \mathsf{e}_x)$. These signatures are also used later to prove that the keys have been output by the authority.

To generate an update token in this scheme, the authority simply executes the key generation procedure of the two-input partially-hiding predicate encryption scheme for the update policy F' to generate the corresponding functional key $\mathsf{sk}_{F'}$. Afterwards, this key is signed by the authority and, together with this signature, output as the update token.

If a sender now wants to generate a signature for a receiver, it uses its secret key $(\mathsf{sk}_S, \mathsf{e}_S)$ together with the public key of the receiver $\mathsf{pk}_R \coloneqq (\mathsf{vk}_R, \mathsf{ct}_R)$ and the functional key $\mathsf{sk}_{F'}$ that is part of the current token. In the first step, the sender verifies the signature of the receiver that is contained in its public key as well as the signature that has been generated for the update token $\mathsf{sk}_{F'}$ to check their authenticity. Afterwards, the sender executes the decryption procedure of the two-input partially-hiding predicate encryption scheme using the functional key $\mathsf{sk}_{F'}$, the ciphertext of the receiver ct_R, and its own encoding e_S to check if the sender and the receiver fulfill the current policy. If this is the case, the sender generates a non-interactive zero-knowledge proof π over the output of the decryption procedure using the information of the receiver's public key, the update token and its own public and secret key. Here, the information contained in the public keys of the parties is used as part of the statement. The final signature σ, for a message m, is then generated by signing (m, pk_R, π) using the signing key sk of the sender.

[6]A recent work from [ATY23] can be used to instantiate 2-PE from bilinear maps for inner-product predicates, but it does not support (strong) attribute-hiding. To the best of our knowledge, there are no (even one-input) PE schemes for general predicates based on standard assumptions satisfying this strong notion of privacy [GVW15, GKW17, WZ17].

For the verification of a signature, it is then simply required to verify that a signature associated with the public keys of the sender and the receiver, as well as the signature of the token for which it has been generated. If these verifications succeed, then the information inside the public keys and the token can be used to verify the proof π of the signature. In the last step, the signature σ generated over the proof and the message $(m, \mathsf{pk}_R, \sigma)$ is verified using the verification key of the sender vk_S. If all of these verification checks succeed, then the signature is deemed valid. We describe the formal scheme in Figure 8.

4.2.1 Correctness and Security

The correctness of the scheme follows directly from the correctness of the underlying schemes: DS, NIZK and that of 2-PHPE. For security, unforgeability follows from unforgeability of the signature schemes and the (knowledge) soundness of the NIZK proof. Further, the attribute-hiding of UPCS follows from the zero-knowledge property of NIZK, as well as the partially attribute-hiding of 2-PHPE. We present the formal proofs below. We note in passing that when switching to a model where we cannot erase the randomness during proof generation, we obtain an analogous result when switching to a NIZK that supports adaptive corruptions [GOS06].

Unforgeability

Now, we prove unforgeability of the scheme presented above.

Theorem 4.3. *Let* $\mathsf{DS}_{\mathsf{pub}} = (\mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify})$*,* $\mathsf{DS}_{\mathsf{tok}} = (\mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify})$*,* $\mathsf{DS}_{\mathsf{priv}} = (\mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify})$ *and* $\mathsf{DS}_{\mathsf{P}} = (\mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{P}}.\mathsf{Verify})$ *be EUF-CMA secure signatures schemes, and let* $\mathsf{NIZK} = (\mathsf{NIZK}.\mathsf{Setup}, \mathsf{NIZK}.\mathsf{Prove}, \mathsf{NIZK}.\mathsf{Verify})$ *be an extractable proof system, then the construction* $\mathsf{UPCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figure 8, is existentially unforgeable.*

Proof. The proof of unforgeability for this construction proceeds very similar to the unforgeability proof of the PCS scheme presented [BMW21]. The main difference being that we do not need to rely on strong unforgeability since, in this scheme, the (unique) party identifier P_i is part of the public key. Additionally, we need to bound an event that prevents an adversary from producing a valid forgery w.r.t. a policy update.

Consider the unforgeability experiment for which we define the following two events:

- Event $\mathsf{KeyForge}_{\mathcal{A}}$: The adversary $\mathcal{A}$ terminates with output $(\mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ where the public key pk_S (resp. pk_R) does not belong to any initialized party P_S (resp. P_R) (by means of an invocation $(\textsc{KeyGen}, P_S, x_S)$, (resp. $(\textsc{KeyGen}, P_R, x_R)$)).
- Event $\mathsf{KeyColl}_{\mathcal{A}}$: The adversary terminates and it holds that there are two parties P_i and P_j where $i \neq j$ such that for the corresponding public keys $\mathsf{pk}_i = (\mathsf{vk}_i, \cdot, \cdot, \cdot)$ and $\mathsf{pk}_j = (\mathsf{vk}_j, \cdot, \cdot, \cdot)$ it holds that $\mathsf{vk}_i = \mathsf{vk}_j$.

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

- Event $\mathsf{WIN1}_{\mathcal{A}}$: The adversary generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where the public key pk is associated with some initialized party P_S that is not corrupted and was never invoked on input $(\textsc{Sign}, P_S, \mathsf{tok}, \mathsf{pk}_R, m)$.

```
$\underline{\mathsf{Setup}(1^\lambda, F_{\mathsf{init}} \in \mathcal{F}_\lambda):}$
$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
$\mathsf{msk_{PE}} \leftarrow \mathsf{2\text{-}PHPE.Setup}(1^\lambda, \mathcal{F}_\lambda)$
$\mathsf{sk}_{F_{\mathsf{init}}} \leftarrow \mathsf{2\text{-}PHPE.KeyGen}(\mathsf{msk_{PE}}, F_{\mathsf{init}})$
$(\mathsf{vk_{pub}}, \mathsf{sk_{pub}}) \leftarrow \mathsf{DS_{pub}.Setup}(1^\lambda)$
$(\mathsf{vk_{priv}}, \mathsf{sk_{priv}}) \leftarrow \mathsf{DS_{priv}.Setup}(1^\lambda)$
$(\mathsf{vk_{tok}}, \mathsf{sk_{tok}}) \leftarrow \mathsf{DS_{tok}.Setup}(1^\lambda)$
$\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS_{tok}.Sign}(\mathsf{sk_{tok}}, (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}))$
$\mathsf{mpk} := (\mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}}, \mathsf{vk_{tok}})$
$\mathsf{msk} := (\mathsf{msk_{PE}}, \mathsf{sk_{pub}}, \mathsf{sk_{priv}}, \mathsf{sk_{tok}})$
$\mathsf{tok_{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$
Return $(\mathsf{mpk}, \mathsf{msk}), \mathsf{tok_{init}}$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, P_i, x):}$
Parse $\mathsf{msk} = (\mathsf{msk_{PE}}, \mathsf{sk_{pub}}, \mathsf{sk_{priv}}, \mathsf{sk_{tok}})$
$(\mathsf{vk_P}, \mathsf{sk_P}) \leftarrow \mathsf{DS_P.Setup}(1^\lambda)$
$\mathsf{e}_x \leftarrow \mathsf{2\text{-}PHPE.Encode}(\mathsf{msk_{PE}}, x)$
$\mathsf{ct} \leftarrow \mathsf{2\text{-}PHPE.Encrypt}(\mathsf{msk_{PE}}, x)$
$\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS_{pub}.Sign}(\mathsf{sk_{pub}}, (\mathsf{vk_P}, \mathsf{ct}, P_i))$
$\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS_{priv}.Sign}(\mathsf{sk_{priv}}, (\mathsf{vk_P}, \mathsf{e}_x))$
$\mathsf{pk} := (\mathsf{vk_P}, \mathsf{ct}, P_i, \sigma_{\mathsf{pub}})$
$\mathsf{sk} := (\mathsf{vk_P}, \mathsf{sk_P}, \mathsf{e}_x, \sigma_{\mathsf{priv}})$
Return $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{PolUpd}(\mathsf{mpk}, \mathsf{msk}, F'):}$
Parse $\mathsf{mpk} := (\mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}})$
        $\mathsf{msk} = (\mathsf{msk_{PE}}, \mathsf{sk_{pub}}, \mathsf{sk_{priv}})$
$\mathsf{sk}_{F'} \leftarrow \mathsf{2\text{-}PHPE.KeyGen}(\mathsf{msk_{PE}}, F')$
$\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS_{tok}.Sign}(\mathsf{sk_{tok}}, (F', \mathsf{sk}_{F'}))$
$\mathsf{tok}' := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$
Return $\mathsf{tok}'$

$\underline{\mathsf{Sign}(\mathsf{mpk}, \mathsf{tok}, \mathsf{sk}, \mathsf{pk}_R, m):}$
Parse $\mathsf{mpk} = (\mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}}, \mathsf{vk_{tok}})$,
        $\mathsf{tok} = (F', \mathsf{sk}_{F'}, \sigma_{\mathsf{tok}})$,
        $\mathsf{sk} = (\mathsf{vk}_S, \mathsf{sk}_S, \mathsf{e}_x, \sigma_{\mathsf{priv}})$,
        $\mathsf{pk}_R = (\mathsf{vk}_R, \mathsf{ct}_R, P_R, \sigma_{\mathsf{pub}})$
If $\mathsf{DS_{pub}.Verify}(\mathsf{vk_{pub}}, (\mathsf{vk}_R, \mathsf{ct}_R, P_R), \sigma^R_{\mathsf{pub}}) = 0$
or $\mathsf{DS_{tok}.Verify}(\mathsf{vk_{tok}}, (F', \mathsf{sk}_{F'}), \sigma_{\mathsf{tok}}) = 0$,
    return $\bot$,
$\pi \leftarrow \mathsf{Prove}(\mathsf{CRS}, (\mathsf{vk_{priv}}, \mathsf{sk}_{F'}, \mathsf{vk}_S, \mathsf{ct}_R), (\mathsf{e}_x, \sigma_{\mathsf{priv}}))$
where the NIZK relation is defined in Figure 9.
Erase the randomness used for computing $\pi$.
$\sigma' \leftarrow \mathsf{DS_P.Sign}(\mathsf{sk}_S, (m, \mathsf{pk}_R, \pi))$
Return $(m, \mathsf{pk}_R, \sigma := (\pi, \sigma'))$

$\underline{\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma):}$
Parse $\mathsf{mpk} = (\mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}}, \mathsf{vk_{tok}})$
        $\mathsf{tok} = (F', \mathsf{sk}_{F'}, \sigma_{\mathsf{tok}})$
        $\mathsf{pk}_S = (\mathsf{vk}_S, \mathsf{ct}_S, P_S, \sigma^S_{\mathsf{pub}})$
        $\mathsf{pk}_R = (\mathsf{vk}_R, \mathsf{ct}_R, P_R, \sigma^R_{\mathsf{pub}})$
        $\sigma = (\pi, \sigma')$
(Return 0 if parsing fails or $\sigma = \bot$)
Return $\mathsf{Verify}(\mathsf{vk_{tok}}, (F', \mathsf{sk}_{F'}), \sigma_{\mathsf{tok}})$
    $\wedge \mathsf{Verify}(\mathsf{vk_{pub}}, (\mathsf{vk}_R, \mathsf{ct}_R), \sigma^R_{\mathsf{pub}})$
    $\wedge \mathsf{Verify}(\mathsf{vk_{pub}}, (\mathsf{vk}_S, \mathsf{ct}_S), \sigma^S_{\mathsf{pub}})$
    $\wedge \mathsf{Verify}(\mathsf{CRS}, (\mathsf{vk_{priv}}, \mathsf{vk}_S, \mathsf{ct}_R), \pi)$
    $\wedge \mathsf{Verify}(\mathsf{vk}_S, (m, \mathsf{pk}_R, \pi), \sigma')$
```

Figure 8: Our non-interactive updatable policy-compliant signature scheme.

Relation R_{ZK}:

Instance: $x = (\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F'}, \mathsf{vk}_S, \mathsf{ct}_R)$

Witness: $w = (\mathsf{e}_x, \sigma_{\mathsf{priv}})$

$R_{\mathrm{ZK}}(x, w) = 1$ if and only if:

$\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}_S, \mathsf{e}_x), \sigma_{\mathsf{priv}}) = 1$ and $\text{2-PHPE}.\mathsf{Dec}(\mathsf{sk}_F, (\mathsf{e}_x, \mathsf{ct}_R)) = 1$

Figure 9: Relation used for the non-interactive UPCS scheme in Figure 8.

- Event $\mathsf{WIN2}_{\mathcal{A}}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where the update token tok^* is not associated with a policy F^* that has never been an input to the authority $(\textsc{Update}, F^*)$. We also denote this event as $\mathsf{PolicyForge}_{\mathcal{A}}$.
- Event $\mathsf{WIN3}_{\mathcal{A}}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where there is a policy F' such that tok is the output by $\mathfrak{A}$ on input $(\textsc{Update}, F')$, and there are attributes x_S and x_R associated to some initialized parties P_S and P_R with public keys pk_S and pk_R, respectively (as the result of inputs $(\textsc{KeyGen}, P_S, x_S)$ and $(\textsc{KeyGen}, P_R, x_R)$ to $\mathfrak{A}$), such that $F'(x_S, x_R) = 0$.

By Lemmata 4.4 and 4.5, we obtain

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

for adversaries $\mathcal{B}_1$ and $\mathcal{B}'_2$ which are constructed based on $\mathcal{A}$ and have roughly the same efficiency as $\mathcal{A}$ and where q are the number of key generation queries $\textsc{KeyGen}$.

Finally, we obtain by Lemmata 4.6 to 4.8 that

$$\begin{aligned}
&\Pr[\mathsf{WIN1}_{\mathcal{A}}] \;\; \le q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{P}},\mathcal{B}''_2}(\lambda),\\
&\Pr[\mathsf{WIN2}_{\mathcal{A}} / \ \mathsf{PolicyForge}_{\mathcal{A}}] \le \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{tok}},\mathcal{B}'_1}(\lambda) \text{ and}\\
&\Pr[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]\\
&\qquad\qquad \le \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK},\mathcal{B}'},
\end{aligned}$$

where the adversaries $\mathcal{B}'_1, \mathcal{B}''_2$, $\mathcal{B}_3$, $\mathcal{B}_4$, 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

$$\begin{aligned}
\Pr[\mathsf{WIN}_{\mathcal{A}}] \le\ & \Pr[\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}]\\
& + \Pr[\mathsf{WIN}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]\\
\le\ & \Pr[\mathsf{KeyColl}_{\mathcal{A}}] + \Pr[\mathsf{KeyForge}_{\mathcal{A}}] + \Pr[\mathsf{PolicyForge}_{\mathcal{A}}]\\
& + \Pr[\mathsf{WIN1}_{\mathcal{A}} \ \cap \ \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]\\
& + \underbrace{\Pr[\mathsf{WIN2}_{\mathcal{A}} \ \cap \ \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]}_{=0, \text{ since } \mathsf{WIN2}_{\mathcal{A}} = \mathsf{PolicyForge}_{\mathcal{A}}}\\
& + \Pr[\mathsf{WIN3}_{\mathcal{A}} \ \cap \ \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}].
\end{aligned}$$

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 $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{P}}}$. Therefore, the theorem follows. □

Lemma 4.4. *Consider the unforgeability experiment and let* $\mathsf{KeyForge}_{\mathcal{A}}$ *be defined as above. We construct an adversary* $\mathcal{B}$ *such that* $\Pr[\mathsf{KeyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}}(\lambda)$.

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

In the first step, the adversary $\mathcal{B}$ receives $\mathsf{vk}_{\mathsf{pub}}$ from the forgeability experiment $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{pub}}}$ and the policy F_{init} from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}$ generates $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^\lambda)$, $\mathsf{CRS} \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{msk}_{\mathsf{PE}} \leftarrow \text{2-}\mathsf{PHPE}.\mathsf{Setup}(1^\lambda)$ and sets $\mathsf{mpk} := (\mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$. Furthermore, $\mathcal{B}$ computes $\mathsf{sk}_{F_{\mathsf{init}}} \leftarrow \text{2-}\mathsf{PHPE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{PE}}, F_{\mathsf{init}})$ and $\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}(\mathsf{sk}_{\mathsf{tok}}, (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}))$, sets $\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$ and sends mpk and $\mathsf{tok}_{\mathsf{init}}$ to the adversary $\mathcal{A}$. The adversary $\mathcal{B}$ also initializes the counter $e = 2$.

If the adversary $\mathcal{A}$ asks a policy update query F', the adversary $\mathcal{B}$ computes $\mathsf{sk}_{F'} \leftarrow \text{2-}\mathsf{PHPE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{PE}}, F')$, $\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, (F', \mathsf{sk}_{F'}))$ and outputs $\mathsf{tok}' := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$ to $\mathcal{A}$. Afterwards, it increases e, i.e. $e := e + 1$.

Whenever $\mathcal{A}$ asks a key generation query $(\textsc{KeyGen}, P_i, x)$, the adversary $\mathcal{B}$ samples a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$, computes $\mathsf{e}_x \leftarrow \text{2-}\mathsf{PHPE}.\mathsf{Encode}(\mathsf{msk}_{\mathsf{PE}}, x)$ and $\mathsf{ct} \leftarrow \text{2-}\mathsf{PHPE}.\mathsf{Enc}(\mathsf{msk}_{\mathsf{PE}}, x)$ and generates the signature $\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{e}_x))$. In the next step, the adversary $\mathcal{B}$ submits $(\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, P_i)$ to its signing oracle and receives σ_{pub} as a reply. Then, $\mathcal{B}$ sets $\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, P_i, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{e}_x, \sigma_{\mathsf{priv}})$ and sends pk to $\mathcal{A}$.

For every corruption query $(\textsc{Corrupt}, P_j)$, $\mathcal{B}$ returns sk to $\mathcal{A}$ for $(j, \mathsf{pk}, \mathsf{sk}, x) \in \mathcal{QK}$, and returns $\perp$ if no such entry exists.

If the adversary $\mathcal{A}$ asks a signing query $(\textsc{Sign}, \mathsf{tok}, P_j, \mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, P_R, \sigma^R_{\mathsf{pub}}), m)$, if tok has been output by a previous query $\textsc{Update}$, then the adversary $\mathcal{B}$ executes $\mathsf{Sign}(\mathsf{mpk}, \mathsf{tok}, \mathsf{sk}, \mathsf{pk}_R, m)$ with sk being the secret key associated with P_j and sends the output as a reply to $\mathcal{A}$. If no key for P_j or no token tok 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 $(\mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$, adversary $\mathcal{B}$ considers the set of all queries to $\textsc{Sign}$ and searches for a tuple $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_{\mathsf{tok}}), \mathsf{pk}^*_S := (\mathsf{vk}^*_S, \mathsf{ct}^*_S, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_R, \mathsf{ct}^*_R, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^*)$ that fulfills the condition of the event $\mathsf{KeyForge}_{\mathcal{A}}$, that is, the condition that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}^*_S, \mathsf{pk}^*_R, m^*, \sigma^*) = 1$ and where the public key pk^*_S or pk^*_R does not belong to any initialized party P_S or P_R. If no such tuple exists, the adversary aborts with output $\perp$. Otherwise, at least one of the two message-signature pairs $((\mathsf{vk}^*_S, \mathsf{ct}^*_S, P^*_S), \sigma^*_{\mathsf{pub},S})$ or $((\mathsf{vk}^*_R, \mathsf{ct}^*_R, P^*_R), \sigma^*_{\mathsf{pub},R})$ has to be a forgery of the underlying $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{pub}}}$ experiment w.r.t. public key $\mathsf{vk}_{\mathsf{pub}}$. Without loss of generality, let pk^*_S be the key that has not been output by $\textsc{KeyGen}$ fulfilling the above condition. This is a valid forgery, since either the pair $(\mathsf{vk}^*_S, \mathsf{ct}^*_S, P^*_S)$ has never been queried to the signing oracle of $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{pub}}}$ by $\mathcal{B}$, or, if it has been queried, the result was not $\sigma^*_{\mathsf{pub},S}$ as otherwise the key pk^*_S would have been an answer to $\textsc{KeyGen}$. This concludes the proof. □

Lemma 4.5. *Consider the unforgeability experiment and let* $\mathsf{KeyColl}_{\mathcal{A}}$ *be defined as above. We construct an adversary* $\mathcal{B}$ *such that* $\Pr[\mathsf{KeyColl}_{\mathcal{A}}] \leq q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{P}},\mathcal{B}}(\lambda)$.

Proof. This proof proceeds exactly as the proof of [BMW21, Lemma 4.4]. We recap it here verbatim for completeness.

We observe that in the the execution of the unforgeability experiment 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 signatures for any message m that will successfully verify w.r.t. $\tilde{\mathsf{vk}}$. □

Lemma 4.6. *Consider the unforgeability experiment 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 Corrupt. Formally, the adversary $\mathcal{A}$ outputs $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ in the EUF-CMA$^{\mathsf{UPCS}}$ game such that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where the public key pk is associated with some initialized party P_S that is not corrupted and was never invoked on input $(\textsc{Sign}, P_S, \mathsf{tok}, \mathsf{pk}_R, m)$.

When q denotes the number of key generation queries, 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 party for which a key generation query has been asked and at least one party is honest.

Now, we bound $\Pr[E_k]$: we build an adversary $\mathcal{B}_k$ that simulates EUF-CMA$^{\mathsf{UPCS}}$ towards $\mathcal{A}$ when interacting with the underlying existential unforgeability experiment EUF-CMA$^{\mathsf{DS_P}}$ and show that if $\mathcal{A}$ outputs $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$, 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 $\mathsf{vk}_\mathsf{P}^{\mathsf{chall}}$ from experiment EUF-CMA$^{\mathsf{DS_P}}$ and the policy F_{init} from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}_k$ generates $(\mathsf{vk_{pub}}, \mathsf{sk_{pub}}) \leftarrow \mathsf{DS_{pub}.Setup}(1^\lambda)$, $(\mathsf{vk_{priv}}, \mathsf{sk_{priv}}) \leftarrow \mathsf{DS_{priv}.Setup}(1^\lambda)$, $(\mathsf{vk_{tok}}, \mathsf{sk_{tok}}) \leftarrow \mathsf{DS_{tok}.Setup}(1^\lambda)$, $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and $\mathsf{msk_{PE}} \leftarrow \text{2-}\mathsf{PHPE.Setup}(1^\lambda)$, sets $\mathsf{mpk} := (\mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}})$, computes $\mathsf{sk}_{F_\mathsf{init}} \leftarrow \text{2-}\mathsf{PHPE.KeyGen}(\mathsf{msk_{PE}}, F_\mathsf{init})$ and $\sigma_\mathsf{tok} \leftarrow \mathsf{DS_{tok}.Sign}(\mathsf{sk_{tok}}, (F_\mathsf{init}, \mathsf{sk}_{F_\mathsf{init}}))$, sets $\mathsf{tok_{init}} := (F_\mathsf{init}, \mathsf{sk}_{F_\mathsf{init}}, \sigma_\mathsf{tok})$ and sends mpk and $\mathsf{tok_{init}}$ to the adversary $\mathcal{A}$. The adversary $\mathcal{B}_k$ also initializes the counter $e = 2$.

Whenever the adversary $\mathcal{A}$ asks a policy update query $(\textsc{Update}, F')$, the adversary $\mathcal{B}$ computes $\mathsf{sk}_{F'} \leftarrow \text{2-}\mathsf{PHPE.KeyGen}(\mathsf{msk_{PE}}, F')$, $\sigma'_\mathsf{tok} \leftarrow \mathsf{DS_{tok}.Sign}(\mathsf{sk_{tok}}, (F', \mathsf{sk}_{F'}))$ and outputs $\mathsf{tok}' := (F', \mathsf{sk}_{F'}, \sigma'_\mathsf{tok})$ to $\mathcal{A}$. Afterwards, it increases e, i.e., $e := e + 1$.

For a key generation query $(\textsc{KeyGen}, P_i, x)$, 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 $(\mathsf{vk_P}, \mathsf{sk_P}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$ and computes $\mathsf{e}_x \leftarrow \text{2-}\mathsf{PHPE.Encode}(\mathsf{msk_{PE}}, x)$ and $\mathsf{ct} \leftarrow \text{2-}\mathsf{PHPE.Enc}(\mathsf{msk_{PE}}, x)$. In the next step, the adversary $\mathcal{B}_k$ generates the signatures σ_pub and σ_priv, where $\sigma_\mathsf{pub} \leftarrow \mathsf{Sign}(\mathsf{sk_{pub}}, (\mathsf{vk_P}, \mathsf{ct}, P_i))$ and $\sigma_\mathsf{priv} \leftarrow \mathsf{Sign}(\mathsf{sk_{priv}}, (\mathsf{vk_P}, \mathsf{e}_x))$. Then, $\mathcal{B}_k$ sets $\mathsf{pk} := (\mathsf{vk_P}, \mathsf{ct}, P_i, \sigma_\mathsf{pub})$ and $\mathsf{sk} := (\mathsf{vk_P}, \mathsf{sk_P}, \mathsf{e}_x, \sigma_\mathsf{priv})$ and sends pk to $\mathcal{A}$.

If the query is the k'th key generation query, i.e. $(\textsc{KeyGen}, P_k, x)$, $\mathcal{B}$ 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 $\mathsf{pk} := (\mathsf{vk}_\mathsf{P}^{\mathsf{chall}}, \mathsf{ct}, P_k, \sigma_\mathsf{pub})$ and $\mathsf{sk} := (\mathsf{vk}_\mathsf{P}^{\mathsf{chall}}, \cdot, \mathsf{e}_x, \sigma_\mathsf{priv})$ and sends pk to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks a corruption query $(\textsc{Corrupt}, P_j)$, the adversary $\mathcal{B}_k$ finds the secret key sk of the party P_j. If no such key exists, the adversary $\mathcal{B}_k$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk to $\mathcal{A}$. In the case that $\mathcal{A}$ asks a corruption query for the k'th party P_k, 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 signing query $(\textsc{Sign}, \mathsf{tok} := (F', \mathsf{sk}_{F'}, \sigma_{\mathsf{tok}}), P_j, \mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, P_R, \sigma_{\mathsf{pub},R}), m)$, then the adversary $\mathcal{B}_k$ checks that a key has been generated for party P_j and returns $\perp$ if this is not the case. Now, we distinguish between two cases: first the signature is requested for the k'th party, i.e. $j = k$, and, second, the signature is not requested for the k'th public key, i.e. $j \neq k$. In the first case, $\mathcal{B}_k$ checks that $\mathsf{PE.Dec}(\mathsf{sk}_{F'}, \mathsf{e}_S, \mathsf{ct}_R) = 1$, computes $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, (\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F'}, \mathsf{vk}_S, \mathsf{ct}_R), (\mathsf{e}_S, \sigma_{\mathsf{priv}}))$ with $\mathsf{sk}_{F'}$ being the key associated with the token tok and $\mathsf{vk}_S, \mathsf{e}_S$ and σ_{priv} being part of the secret key of P_k, submits (m, pk_R, π) to the signing oracle of EUF-CMA$^{\mathsf{DS_P}}$ 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 the secret key sk of the party P_j and sends the resulting signature σ as a reply to $\mathcal{A}$.

Finally, when $\mathcal{A}$ terminates with output $(\mathsf{tok}^*, \mathsf{pk}^*_S := (\mathsf{vk}^*_S, \mathsf{ct}^*_S, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_R, \mathsf{ct}^*_R, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^*, \sigma'))$ check the conditions of E_k, i.e., $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}^*_S, \mathsf{pk}^*_R, m^*, \sigma^*) = 1$ where the public key pk^* is associated with some initialized party P_S that is not corrupted and was never invoked on input $(\textsc{Sign}, P_S, \mathsf{tok}^*, \mathsf{pk}^*_R, m^*)$, and verify that pk^*_S is indeed the public key associated to party P_k, in which case $\mathsf{pk}^*_S = (\mathsf{vk}^{\mathsf{chall}}_{\mathsf{P}}, \dots)$. In this case the adversary $\mathcal{B}_k$ outputs $((m^*, \mathsf{pk}^*_R, \pi^*), \sigma')$ as its forgery to the underlying EUF-CMA$^{\mathsf{DS_P}}$ 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$^{\mathsf{DS_P}}$. By definition of E_k, we must in particular have a valid signature for $(m^*, \mathsf{pk}^*_R, \pi^*)$ w.r.t. $\mathsf{vk}^{\mathsf{chall}}_{\mathsf{P}}$ specified in pk^*_S. If m^* was never part of any signing query by $\mathcal{A}$ for party P_k as a signer, then the output of $\mathcal{B}_k$ in experiment EUF-CMA$^{\mathsf{DS_P}}$ 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 q independently and honestly sampled public keys (and correctly computed signatures thereof) and the emulated oracle calls execute the scheme as in the unforgeability experiment, thus the definition of event E_k applies directly to the emulation and the probability is the same as in an execution of the unforgeability experiment.

Therefore, the probability that $\mathcal{B}_k$ terminates with a valid forgery is $\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS_P},\mathcal{B}_k}(\lambda) = \Pr[E_k]$. The proof concludes by sampling $k \leftarrow [q]$ and executing $\mathcal{B}_k$. $\square$

Lemma 4.7. *Consider the unforgeability experiment and let* $\mathsf{WIN2}_{\mathcal{A}}/\mathsf{PolicyForge}_{\mathcal{A}}$ *be defined as above. We construct an adversary* $\mathcal{B}$ *such that* $\Pr[\mathsf{WIN2}_{\mathcal{A}}/\mathsf{PolicyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS_{tok}},\mathcal{B}}(\lambda)$.

Proof. We build an adversary $\mathcal{B}$ that simulates EUF-CMA$^{\mathsf{UPCS}}$ towards $\mathcal{A}$ when interacting with the underlying unforgeability experiment EUF-CMA$^{\mathsf{DS_{tok}}}$ and show that if $\mathcal{A}$ outputs $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ as described in the event $\mathsf{WIN2}_{\mathcal{A}}/\mathsf{PolicyForge}_{\mathcal{A}}$, it can be used as a forgery in the EUF-CMA$^{\mathsf{DS_{tok}}}$ experiment.

In the first step, the adversary $\mathcal{B}$ receives $\mathsf{vk_{tok}}$ from the forgeability experiment EUF-CMA$^{\mathsf{DS_{tok}}}$ and the policy F_{init} from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}$ generates $(\mathsf{vk_{pub}}, \mathsf{sk_{pub}}) \leftarrow \mathsf{DS_{priv}.Setup}(1^\lambda)$, $(\mathsf{vk_{priv}}, \mathsf{sk_{priv}}) \leftarrow \mathsf{DS_{tok}.Setup}(1^\lambda)$, $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and $\mathsf{msk_{PE}} \leftarrow$ 2-PHPE.$\mathsf{Setup}(1^\lambda)$ and sets $\mathsf{mpk} := (\mathsf{CRS}, \mathsf{vk_{pub}}, \mathsf{vk_{priv}}, \mathsf{vk_{tok}})$. Furthermore, $\mathcal{B}$ computes $\mathsf{sk}_{F_{\mathsf{init}}} \leftarrow$ 2-PHPE.KeyGen $(\mathsf{msk_{PE}}, F_{\mathsf{init}})$ and queries the signing oracle of its underlying challenger using $(F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}})$ to obtain the signature σ_{tok}. Afterwards, it sets $\mathsf{tok_{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$ and sends mpk and $\mathsf{tok_{init}}$ to the adversary $\mathcal{A}$. The adversary $\mathcal{B}$ also initializes the counter $e = 2$.

If the adversary $\mathcal{A}$ asks a policy update query $(\textsc{Update}, F')$, the adversary $\mathcal{B}$ computes $\mathsf{sk}_{F'} \leftarrow$ 2-PHPE.KeyGen$(\mathsf{msk}_{\mathsf{PE}}, F')$ and sends $(F', \mathsf{sk}_{F'})$ to the singing oracle of its underlying challenger to obtain σ'_{tok} and outputs $\mathsf{tok}' := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$ to $\mathcal{A}$. Afterwards, it increases e, i.e. $e := e + 1$.

Whenever $\mathcal{A}$ asks a key generation query $(\textsc{KeyGen}, P_i, x)$, then the adversary $\mathcal{B}$ samples a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$, computes $\mathsf{e}_x \leftarrow$ 2-PHPE.Encode$(\mathsf{msk}_{\mathsf{PE}}, x)$ and $\mathsf{ct} \leftarrow$ 2-PHPE.Enc$(\mathsf{msk}_{\mathsf{PE}}, x)$ and generates the signatures $\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, P_i))$ and $\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{e}_x))$. Afterwards, $\mathcal{B}$ sets $\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, P_i, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{e}_x, \sigma_{\mathsf{priv}})$ and sends pk to $\mathcal{A}$.

For every corruption query $(\textsc{Corrupt}, P_j)$, $\mathcal{B}$ finds the secret key sk of the party P_j and sends it to $\mathcal{A}$. If no key for P_j exists, it returns $\perp$.

If the adversary $\mathcal{A}$ asks a signing query $(\textsc{Sign}, \mathsf{tok}, P_j, \mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, P_R, \sigma^R_{\mathsf{pub}}), m)$, if the token tok has been previously been used as the answer to an $\textsc{Update}$ query, the adversary $\mathcal{B}$ executes $\mathsf{Sign}(\mathsf{mpk}, \mathsf{tok}, \mathsf{sk}, \mathsf{pk}_R, m)$ with sk being the secret key associated with party P_j and sends the output as a reply to $\mathcal{A}$. If no key for party P_j or no token tok has previously been output by $\textsc{Update}$, output $\perp$.

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 $(\mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$, adversary $\mathcal{B}$ considers the set of all queries to $\textsc{Sign}$ and searches for a tuple $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_{\mathsf{tok}}), \mathsf{pk}^*_S := (\mathsf{vk}^*_S, \mathsf{ct}^*_S, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_R, \mathsf{ct}^*_R, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^*)$ that fulfills the condition of the event $\mathsf{WIN2}_{\mathcal{A}}$, that is, the condition that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}^*_S, \mathsf{pk}^*_R, m^*, \sigma^*) = 1$ and where the update token tok^* has never been computed by the authority on any input $(\textsc{Update}, F^*)$. If no such tuple exists, the adversary aborts with output $\perp$. Otherwise, $((F^*, \mathsf{sk}_{F^*}), \sigma^*_{\mathsf{tok}})$ has to be a forgery of the underlying EUF-CMA$^{\mathsf{DS}_{\mathsf{tok}}}$ experiment w.r.t. policy F^*. This is a valid forgery, since either the policy F^* has never been queried to the signing oracle of EUF-CMA$^{\mathsf{DS}_{\mathsf{tok}}}$ by $\mathcal{B}$, or, if it has been queried, the reply was not σ^*_{tok} as otherwise the token tok^* would have been an answer to a policy update query $\textsc{Update}$. This concludes the proof. ☐

Lemma 4.8. *Consider the unforgeability experiment and let* $\mathsf{WIN3}_{\mathcal{A}}$ *be defined as above. We can construct adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *and a distinguisher* $\mathcal{B}'$ *such that*

$$\Pr[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}] \leq \mathsf{Adv}^{\mathrm{CRS}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\mathrm{EUF\text{-}CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda).$$

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

$$\begin{aligned}\Pr[&\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}] \\ &\leq \Pr_{\mathrm{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right] + \mathsf{Adv}^{\mathrm{CRS}}_{\mathsf{NIZK},\mathcal{B}'},\end{aligned}$$

where $\Pr_{\mathrm{Hyb}}[.]$ makes it explicit that this probability is taken w.r.t. experiment EUF-CMA$^{\mathsf{UPCS}}_{\mathrm{Hyb}}$. 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 $\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}$ occurs (which is computable by $\mathcal{B}'$ that manages all key-sets).

We build an adversary $\mathcal{B}_1$ that simulates $\text{EUF-CMA}^{\mathsf{UPCS}}_{\text{Hyb}}$ towards $\mathcal{A}$ when interacting with the underlying $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{priv}}}$ experiment. We show that if $\mathcal{A}$ outputs $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ 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 $\mathsf{Fail}_{\text{ext}}$ 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_{init} from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}_1$ generates $\mathsf{CRS} \leftarrow E_1(1^\lambda)$, $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{msk}_{\mathsf{PE}} \leftarrow \mathsf{PE.Setup}(1^\lambda)$, sets $\mathsf{mpk} := (\mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$. Furthermore, $\mathcal{B}$ computes $\mathsf{sk}_{F_{\mathsf{init}}} \leftarrow \text{2-}\mathsf{PHPE.KeyGen}(\mathsf{msk}_{\mathsf{PE}}, F_{\mathsf{init}})$ and generates $\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}(\mathsf{sk}_{\mathsf{tok}}, (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}))$. Afterwards, it sets $\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$ and sends mpk and $\mathsf{tok}_{\mathsf{init}}$ to the adversary $\mathcal{A}$. The adversary $\mathcal{B}$ also initializes the counter $e = 2$.

If the adversary $\mathcal{A}$ asks a policy update query $(\textsc{Update}, F')$, the adversary $\mathcal{B}$ computes $\mathsf{sk}_{F'} \leftarrow \text{2-}\mathsf{PHPE.KeyGen}(\mathsf{msk}_{\mathsf{PE}}, F')$, generates $\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}(\mathsf{sk}_{\mathsf{tok}}, (F', \mathsf{sk}_{F'}))$ and outputs $\mathsf{tok}' := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$ to $\mathcal{A}$. Afterwards, it increases e, i.e., $e := e + 1$.

For a key generation query $(\textsc{KeyGen}, P_i, x)$, $\mathcal{B}_1$ samples a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$, computes $\mathsf{e}_x \leftarrow \text{2-}\mathsf{PHPE.Encode}(\mathsf{msk}_{\mathsf{PE}}, x)$ and $\mathsf{ct} \leftarrow \text{2-}\mathsf{PHPE.Enc}(\mathsf{msk}_{\mathsf{PE}}, x)$. In the next step, the adversary $\mathcal{B}_1$ generates the signatures $\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, P_i))$ and submits $(\mathsf{vk}_{\mathsf{P}}, \mathsf{e}_x)$ 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 $\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{ct}, P_i, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{e}_x, \sigma_{\mathsf{priv}})$ 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 $(\textsc{Corrupt}, P_j)$, the adversary $\mathcal{B}_1$ searches for the key sk that is associated with P_j. If no such entry exists, the adversary $\mathcal{B}_1$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk to $\mathcal{A}$.

If the adversary $\mathcal{A}$ asks a signing query $(\textsc{Sign}, \mathsf{tok}, P_j, \mathsf{pk}_R := (\mathsf{vk}_R, \mathsf{ct}_R, P_R, \sigma^R_{\mathsf{pub}}), m)$, then the adversary $\mathcal{B}_1$ checks that a key has been generated for P_j and that pk_R has been the reply to a previous key generation query $\textsc{KeyGen}$. If this is not the case the adversary $\mathcal{B}_1$ aborts. Otherwise, it continues with the execution of Sign, using the key sk of party P_j. Finally, $\mathcal{B}_1$ sends the resulting signature σ, containing π, as a reply to $\mathcal{A}$.

When $\mathcal{A}$ terminates with $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_{\mathsf{tok}}), \mathsf{pk}^*_S := (\mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_S, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_{\mathsf{P},R}, \mathsf{ct}^*_R, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}_1$ 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 $\mathsf{WIN3}_{\mathcal{A}}$ hold and in case this is true, it first calls $(\mathsf{e}^*_{x,S}, \sigma^*_{\mathsf{priv},S}) \leftarrow E_2(\mathsf{CRS}, (\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F^*}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_R), \pi^*)$ and checks whether $(x := (\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F^*}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_R), w := (\mathsf{e}^*_{x,S}, \sigma^*_{\mathsf{priv},S})) \in R_{\text{ZK}}$ (which is efficiently checkable) and if this is the case, it submits $((\mathsf{vk}^*_{\mathsf{P},S}, \mathsf{e}^*_{x,S}), \sigma^*_{\mathsf{priv},S})$ as a forgery to the underlying experiment $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{priv}}}$. If $(x, w) \notin R_{\text{ZK}}$ then it aborts with failure event $\mathsf{Fail}_{\text{ext}}$.

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 events in this emulation are defined as in the experiment $\text{EUF-CMA}^{\mathsf{UPCS}}_{\text{Hyb}}$ with respectively the same probabilities.

Now, we analyze the final forgery output of a run of $\mathcal{B}_1$ which does not abort. We observe that, in this case, all signature verification keys are unique and that all keys can be uniquely associated with some attributes as all keys, including the output pk^*_S and pk^*_R of $\mathcal{A}$, have been previously been an answer to a query $\textsc{KeyGen}$. Therefore, there are parties P_j and P_k such that pk^*_S is the key of P_j and pk^*_R is the key of P_k and where the keys contain vk_i and ct_i and are associated with x_i for $i \in \{j, k\}$. Let us fix these two indices. Furthermore, we can assume that $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F^*}, \mathsf{vk}_j, \mathsf{ct}_k), \pi^*) = 1$ as otherwise, $\mathsf{WIN3}$ does not hold. Additionally, we know that $\mathsf{e}^*_{x,S}$ is a correct witness, in particular, $\mathsf{PE.Dec}(\mathsf{sk}_{F^*}, (\mathsf{e}^*_{x,S}, \mathsf{ct}_k)) = 1$. 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 keys of party P_j $(\mathsf{pk}_j, \mathsf{sk}_j = (\cdot, \cdot, \mathsf{e}_{x_j}, \sigma_{\mathsf{priv},j}))$ specify e_{x_j} for which $\mathsf{PE.Dec}(\mathsf{sk}_{F^*}, (\mathsf{e}_{x_j}, \mathsf{ct}_k)) = 0$, and thus $\mathsf{e}_{x_j} \neq \mathsf{e}^*_{x,S}$. Since, by uniqueness, $\mathcal{B}_1$ has only submitted the query $(\mathsf{vk}_j, \mathsf{e}_{x_j})$ to the signing oracle of EUF-CMA$^{\mathsf{DS}_{\mathsf{priv}}}$, the pair $((\mathsf{vk}^*_{\mathsf{P},S} = \mathsf{vk}_j, \mathsf{e}^*_{x,S} \neq \mathsf{e}_{x_j}), \sigma^*_{\mathsf{priv},S})$ 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

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1} = \Pr_{\text{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\text{ext}}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right].$$

This results in

$$\Pr_{\text{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right] = \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1} + \underbrace{\Pr_{\text{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \mathsf{Fail}_{\text{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right]}_{\leq \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}}.$$

It is straightforward to obtain an adversary $\mathcal{B}_2$ (based on $\mathcal{A}$) which has an advantage $\mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_2} = \Pr_{\text{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \mathsf{Fail}_{\text{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right]$. 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 $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_{\mathsf{tok}}), \mathsf{pk}^*_S := (\mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_S, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_{\mathsf{P},R}, \mathsf{ct}^*_R, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}_2$ behaves as $\mathcal{B}_1$ but does not execute the final steps running the extractor, instead, it just outputs $(x := (\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F^*}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{ct}^*_R), \pi^*)$ in case the conditions of $\mathsf{WIN3}$ are satisfied (note that the extractor is run as part of the experiment in Definition 2.6). As above, the emulation towards $\mathcal{A}$ is perfect until the point where $\mathcal{B}_2$ 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 $\mathsf{WIN3}$) but the extraction produces a witness w such that $(x, w) \notin R_{\text{ZK}}$. Therefore, the statement follows. □

Attribute-Hiding

After proving the unforgeability of our construction, we proceed to proving the attribute-hiding property.

Theorem 4.9. *Let* $\mathsf{2\text{-}PHPE} = (\mathsf{2\text{-}PHPE.Setup}, \mathsf{2\text{-}PHPE.KeyGen}, \mathsf{2\text{-}PHPE.Encode}, \mathsf{2\text{-}PHPE.Encrypt}, \mathsf{2\text{-}PHPE.Dec})$ *be a two-party partially-hiding 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 Figure 9) and let* $\mathsf{DS}_{\mathsf{pub}} = (\mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify})$ *and* $\mathsf{DS}_{\mathsf{tok}} = (\mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify})$ *be unforgeable signature schemes, then the construction* $\mathsf{UPCS} = (\mathsf{Setup}, \mathsf{PolUpd}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, *defined in Figure 8, is attribute-hiding.*

Proof. The proof of this theorem proceeds very similar to the proof of the PCS scheme in [BMW21]. Many parts of this proof are taken verbatim from the proof of [BMW21, Theorem 4.7].

To prove this statement, we use a hybrid argument with the following games:

Game G_0: This game is defined as the attribute-hiding game for $b = 0$.

Game G_1: In this game, we change the behavior of the sign query $\textsc{Sign}$ and define a modified sign query $\textsc{Sign}'$. The query $\textsc{Sign}'$ is defined as $\textsc{Sign}$ with the difference that it only answers queries for tokens that have previously been output by a policy update query $\textsc{Update}$ and receiver keys that have previously been output by a key generation query $\textsc{Test-KeyGen}$, i.e., for a query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ to P_j with tok_e not previously been output by a $\textsc{Update}$ query or pk_i not previously been output by a $\textsc{Test-KeyGen}$ query, the sign query $\textsc{Sign}'$ replies with $\perp$. The transition from G_0 to G_1 is justified by the same reasoning as we have seen in Lemma 4.4 (bounding the event $\mathsf{KeyForge}_{\mathcal{A}}$) and in Lemma 4.7 (bounding the event $\mathsf{PolicyForge}$).

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 $(\textsc{Sign}', \mathsf{tok}_e, j, \mathsf{pk}_i, m)$ to P_j, we find the query $(\textsc{Update}, F_e)$ that has been answered using tok_e and the attributes $x_{j,0}$ associated with pk_i and $x_{j,0}$ s.t. $F_e(x_{j,0}, x_{i,0}) = 1$ ($x_{i,0}$ is associated with the party P_i, i.e, the party that is associated with the key pk_i) and then we simulate the proof using the NIZK simulator on input the trapdoor and $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F'}, \mathsf{vk}_j, \mathsf{ct}_R)$ where $\mathsf{sk}_{F'}$ is part of the token tok_e (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 $\perp$.

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 which results in a ciphertext ct_i that encrypts $x_{i,1}$ instead of $x_{i,0}$. Similarly, upon PCS signing, we now find the policy update query $(\textsc{Update}, F_e)$ that has been answered using tok_e and check that $F_e(x_{j,1}, x_{i,1}) = 1$ where $x_{j,1}$ and $x_{i,1}$ are the attributes associated with the parties/keys of the query, before simulating the proofs as above. The transition from G_2 to G_3 is justified by the attribute-hiding property of 2-PHPE.

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 4.11, this transition is justified by the zero-knowledge property of NIZK.

Game G_5: This game corresponds to the attribute-hiding game using $b = 1$ as its input. In this game, we change back from signing queries $\textsc{Sign}'$ to queries $\textsc{Sign}$. As in Lemma 4.10, this transition is justified again by the inability to forge updates and public keys.

From the definition of the games it is clear that G_0 corresponds to the attribute-hiding game with $b = 0$ and G_5 corresponds to the attribute-hiding game with $b = 1$. Since it holds that $\mathsf{G}_0 \approx_c \cdots \approx_c \mathsf{G}_5$, the theorem follows. $\square$

Lemma 4.10 (Transition from G_0 to G_1)**.** *Let $\mathsf{DS}_{\mathsf{pub}}$ and $\mathsf{DS}_{\mathsf{tok}}$ be unforgeable signature schemes, then the games G_0 and G_1 are computationally indistinguishable.*

Proof (Sketch). As described above, the difference between the games G_0 and G_1 is that in G_0, $\mathcal{A}$ receives answers to $\textsc{Sign}$ queries and in G_1, $\mathcal{A}$ receives replies to the sign queries $\textsc{Sign}'$, which we informally described above and which is formally defined as:

$(\mathbf{Sign}', \mathsf{tok}_e, \mathsf{sk}_j, \mathsf{pk}_i, m)$: On querying a token tok_e, a (sender) secret key sk_j, a (receiver) public key pk_i, and a message m. if tok_e has been the reply to a query $(\textsc{Update}, F_e)$ and pk_i has been the reply to a query $(\textsc{Test-KeyGen}, P_i, x_{i,0}, x_{i,1})$, then return $\sigma \leftarrow \mathsf{UPCS.Sign}(\mathsf{mpk}, \mathsf{tok}_e, \mathsf{sk}_j, \mathsf{pk}_i, m)$. Otherwise, return $\perp$.

Compared to the $\textsc{Sign}'$ queries, the signing queries $\textsc{Sign}$ does not require the token tok as well as the receiver key pk_i to have been previously output by the challenger, i.e., using an $\textsc{Update}$ and a $\textsc{Test-KeyGen}$ query, to obtain as a reply a valid signature $\sigma \neq \perp$. This is not possible for the $\textsc{Sign}'$ queries where every query using a token tok or a receiver key pk_i that has not been generated by the challenger, i.e., a $\textsc{Update}$ query or a $\textsc{Test-KeyGen}$ query, 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 asks a signing query $\textsc{Sign}$ using a token tok or a receiver key pk_i that has not been previously generated by the challenger, i.e., using a $\textsc{Update}$ or a $\textsc{Test-KeyGen}$ query, 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 either generate a token that verifies with respect to the signature scheme $\mathsf{DS}_{\mathsf{tok}}$ or needs to generate a receiver key that verifies with respect to the signature scheme $\mathsf{DS}_{\mathsf{pub}}$, i.e., it needs to generate a token $\mathsf{tok} := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$ such that $\mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{tok}}, (F', \mathsf{sk}_{F'}), \sigma'_{\mathsf{tok}})$ or a key $\mathsf{pk}' := (\mathsf{vk}', \mathsf{ct}', P', \sigma'_{\mathsf{pub}})$ such that $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}', \mathsf{ct}', P'), \sigma'_{\mathsf{pub}}) = 1$ (note that an honest signer issues a signature string only if the validity of the token and the receiver public key is successfully verified). This means that adversary $\mathcal{A}$ must either generate a policy forgery as captured by the event $\mathsf{PolicyForge}_{\mathcal{A}}$ or a key forgery as captured by the event $\mathsf{KeyForge}_{\mathcal{A}}$ in the proof of Theorem 4.3, and which can be defined and analyzed analogously here (with just minor syntactical changes). Therefore, the event $\mathsf{SignForge}_{\mathcal{A}}$ is bounded by $\mathsf{KeyForge}$ and $\mathsf{PolicyForge}$, i.e., $\Pr[\mathsf{SignForge}_{\mathcal{A}}] \leq \Pr[\mathsf{KeyForge}_{\mathcal{A}}] + \Pr[\mathsf{PolicyForge}_{\mathcal{A}}]$, the analysis of the event $\mathsf{KeyForge}_{\mathcal{A}}$ follows the same reasoning as in Lemma 4.4 and the analysis of the event follows the same reasoning as in Lemma 4.7 to conclude that $\Pr[\mathsf{KeyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}_0}(\lambda)$ and $\Pr[\mathsf{PolicyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{tok}},\mathcal{B}_1}(\lambda)$. This results in the fact that $\Pr[\mathsf{SignForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}_0}(\lambda) + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{tok}},\mathcal{B}_1}$, which proves the lemma. □

Lemma 4.11 (Transition from G_1 to G_2)**.** *Let* NIZK *be a zero-knowledge proof system, then* G_1 *and* G_2 *are computationally indistinguishable.*

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

In the beginning of the reduction, $\mathcal{B}$ receives F_{init} from adversary $\mathcal{A}$ and CRS from the $\mathrm{ZK}^{\mathsf{NIZK}}_{\beta}$ experiment. It generates three signature key pairs $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^{\lambda})$, $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^{\lambda})$ and $(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^{\lambda})$, a two-party partially-hiding predicate encryption master secret key $\mathsf{msk}_{\mathsf{2\text{-}PHPE}} \leftarrow \mathsf{2\text{-}PHPE.Setup}(1^{\lambda})$, sets $(\mathsf{mpk}, \mathsf{msk}) = ((\mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}}), (\mathsf{msk}_{\mathsf{2\text{-}PHPE}}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{tok}}))$ and generates the first token, i.e., it generates $\mathsf{sk}_{F_{\mathsf{init}}} \leftarrow \mathsf{2\text{-}PHPE.KeyGen}(\mathsf{msk}_{\mathsf{2\text{-}PHPE}}, F_{\mathsf{init}})$, signs it $\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}))$ and sets $\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$. Later, $\mathcal{B}$ sends mpk and $\mathsf{tok}_{\mathsf{init}}$ to the adversary $\mathcal{A}$ and also initializes the counter $e = 1$.

Whenever $\mathcal{A}$ submits a policy update query $(\textsc{Update}, F')$, $\mathcal{B}$ increaes e, i.e. $e := e + 1$, and generates $\mathsf{sk}_{F'} \leftarrow \mathsf{2\text{-}PHPE.KeyGen}(\mathsf{msk}_{\mathsf{2\text{-}PHPE}}, F')$ and $\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, (F', \mathsf{sk}_{F'}))$. Afterwards, $\mathcal{B}$ sets $\mathsf{tok}_e := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$ and sends tok_e to $\mathcal{A}$.

For every challenge key generation query $(\textsc{Test-KeyGen}, P_i, x_{i,0}, x_{i,1})$ asked by $\mathcal{A}$, $\mathcal{B}$ generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P_i, x_{i,0})$ and sends pk as a reply to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks a corruption query $(\textsc{Corrupt}, P_j)$, the adversary $\mathcal{B}$ finds the corresponding secret key sk_j of the party P_j. If no key exists for party P_j, the adversary $\mathcal{B}$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk_j to $\mathcal{A}$. Here, we highlight that the adversary is not able to determine if the proof has been honestly generated or simulated since the randomness used to generate it has been deleted by the party P_j afterwards.

For every sign query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ asked by $\mathcal{A}$ to P_j, $\mathcal{B}$ finds the key generation query that has been used to generate pk_i and parses $\mathsf{pk}_i = (\mathsf{vk}_i, \mathsf{ct}_i, P_i, \sigma^i_{\mathsf{pub}})$, it also parses the secret key of P_j as $\mathsf{sk} = (\mathsf{vk}_j, \mathsf{sk}_\mathsf{P}, \mathsf{e}_{x^S_{j,0}}, \sigma^j_{\mathsf{priv}})$. If the token tok_e has not been the reply to a query $(\textsc{Update}, F_e)$ or if $F_e(x^{j,0}_S, x^{i,0}_R) = 0$ where $x_{i,0}$ are the attributes associated with P_i, then the adversary $\mathcal{B}$ outputs $\perp$.[7] Otherwise, $\mathcal{B}$ submits $((\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{F_e}, \mathsf{vk}_j, \mathsf{ct}_i), (\mathsf{e}_{x^S_{j,0}}, \sigma_{\mathsf{priv}}))$ to the prove oracle which replies with π. The adversary $\mathcal{B}$ 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 $\mathcal{B}$ 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 $\mathcal{B}$ is simulating the game G_1 and in the case that the challenger simulates the CRS and the proofs, $\mathcal{B}$ is simulating 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 $\mathsf{2\text{-}PHPE.Dec}(\mathsf{sk}_{F_e}, (\mathsf{e}_{x^S_0}, \mathsf{ct}_R)) = F_e(x^0_S, x^0_R)$ (since all UPCS keys and tokens can be assumed to be honestly generated in this hybrid experiment), and thus we are in fact submitting a valid witness. Thus, if the policy is not satisfied, returning $\perp$ is the correct behavior. This covers the simulation of $\mathsf{G}_{1+\beta}$ and leads to the advantage mentioned in the lemma. □

Lemma 4.12 (Transition from G_2 to G_3)**.** *Let* PE *be an attribute-hiding scheme, then the games* G_2 *and* G_3 *are computationally indistinguishable.*

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

In the beginning of the reduction, $\mathcal{B}$ receives F_{init} from the adversary $\mathcal{A}$. It simulates a CRS, i.e., $(\mathsf{CRS}, \tau) \leftarrow \mathsf{Sim}_1(1^\lambda)$, generates two signature key pairs $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$ and $(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^\lambda)$, sets $\mathsf{mpk} := (\mathsf{CRS}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$ and obtains the initial token by submitting F_{init} to the key generation oracle of the underlying challenger to obtain $\mathsf{sk}_{F_{\mathsf{init}}}$. It then generates $\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}))$ sets $\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$ and sends mpk and $\mathsf{tok}_{\mathsf{init}}$ to $\mathcal{A}$. The adversary $\mathcal{B}$ also initializes $e = 1$.

Whenever $\mathcal{A}$ submits a policy update query $(\textsc{Update}, F')$, check that $F(x'_0, x''_0) = F(x'_1, x''_1)$ for all key generation queries $(\textsc{Test-KeyGen}, P_i, x'_0, x'_1)$ and $(\textsc{Test-KeyGen}, P_i, x''_0, x''_1)$. If this check is unsuccessful, the adversary $\mathcal{B}$ outputs a random bit $\alpha \leftarrow \{0,1\}$ as its guess and aborts.[8] If the check is successful, increase e, i.e., $e := e + 1$, and submit F' to the key generation oracle of the underlying challenger to obtain $\mathsf{sk}_{F'}$ and generate $\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, (F', \mathsf{sk}_{F'}))$. Afterwards, $\mathcal{B}$ sets $\mathsf{tok}_e := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$, and outputs tok_e to $\mathcal{A}$.

For every key generation query $(\textsc{Test-KeyGen}, P_i, x_0, x_1)$ asked by $\mathcal{A}$, $\mathcal{B}$ checks that $F_e(x'_0, x_0) = F_e(x'_1, x_1)$ for all $(\textsc{Test-KeyGen}, P_i, x'_0, x'_1)$ and all queries $(\textsc{Update}, F_e)$. If this check is unsuccessful, the adversary $\mathcal{B}$ outputs a random bit $\alpha \leftarrow \{0,1\}$ as its guess and aborts. If the check is successful, the adversary $\mathcal{B}$ submits the left-or-right challenge query (x_0, x_1) to its own encryption oracle to receive ct as a reply. In the next step, $\mathcal{B}$ samples a digital signature key pair $(\mathsf{vk}_\mathsf{P}, \mathsf{sk}_\mathsf{P}) \leftarrow \mathsf{DS}_\mathsf{P}.\mathsf{Setup}(1^\lambda)$, computes $\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_\mathsf{P}, \mathsf{ct}, P_i))$ and outputs $\mathsf{pk} = (\mathsf{vk}_\mathsf{P}, \mathsf{ct}, P_i, \sigma_{\mathsf{pub}})$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks a corruption query $(\textsc{Corrupt}, P_j)$, $\mathcal{B}$ checks the key generation queries to find the query $(\textsc{Test-KeyGen}, P_j, x_0, x_1)$. If no such query exists, the adversary $\mathcal{B}$ returns $\perp$. If it holds

[7]We again note in passing that for a valid adversary, it must hold that $F(x^S_{j,0}, x^R_{i,0}) = F(x^S_{j,1}, x^R_{i,1})$.

[8]Looking ahead, the output of a random bit happens in cases where the adversary is not valid and thus we do not lose any advantage.

that $x_0 \neq x_1$ or if $F_e(x_0, x'_0) \neq F_e(x_1, x'_1)$ for any $(\textsc{Test-KeyGen}, P_i, x'_0, x'_1)$ and $(\textsc{Update}, F_e)$, then the adversary $\mathcal{B}$ outputs a random bit $\alpha \leftarrow \{0,1\}$ as its guess and aborts (again, this only occurs for an invalid $\mathcal{A}$). Otherwise, $\mathcal{B}$ submits $x \coloneqq x_0 \coloneqq x_1$ to the encoding oracle of the underlying challenger to obtain as a reply e_x. In the next step, $\mathcal{B}$ computes $\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, \mathsf{e}_x))$ and outputs $\mathsf{sk} = (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, \mathsf{e}_x, \sigma_{\mathsf{priv}})$ to $\mathcal{A}$, where sk_{P} is the signature key generated during the key generation oracle query.

For every sign query $(\textsc{Sign}, \mathsf{tok}_e, P_j, \mathsf{pk}_R, m)$ asked by $\mathcal{A}$, $\mathcal{B}$ checks the list key generation queries $(\textsc{Test-KeyGen}, P_i, x_0, x_1)$ to find the query associated with pk_R and the policy update queries $(\textsc{Update}, F_e)$ to find the policy F_e associated with the token tok_e. If no public key pk_R or no token tok_e has been generated, then the adversary $\mathcal{B}$ outputs $\perp$. Otherwise, $\mathcal{B}$ checks, in the next step, that the attributes associated with the public keys pk_S and pk_R fulfill the policy F_e, i.e., it checks that $F_e(x_S^0, x_R^0) = 1$ and $F_e(x_S^1, x_R^1) = 1$. If this is the case, the adversary $\mathcal{B}$ simulates the proof π for the language L (defined in Figure 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 F_e, i.e., $F_e(x_S^0, x_R^0) = 0$ or $F_e(x_S^1, x_R^1) = 0$, the adversary $\mathcal{B}$ outputs $\perp$ to $\mathcal{A}$. In the case that the policy evaluations differ, i.e., it holds that either $F_e(x_S^0, x_R^0) = 0$ and $F_e(x_S^1, x_R^1) = 1$ or $F_e(x_S^0, x_R^0) = 1$ and $F_e(x_S^1, x_R^1) = 0$, $\mathcal{B}$ aborts and outputs a random bit $\alpha \leftarrow \{0,1\}$ as its guess to the underlying challenger (the adversary $\mathcal{A}$ is invalid).

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

We need to argue that the adversary $\mathcal{B}$ is a valid adversary with respect to the $\mathrm{AH}_\beta^{\mathsf{2\text{-}PHPE}}$ experiment if the adversary $\mathcal{A}$ fulfills all the checks described above, i.e., is a valid adversary in the $\mathsf{G}_{2+\beta}$ $(\mathrm{AH}_\beta^{\mathsf{UPCS}})$ game. One of the validity requirements above (and in the definition of UPCS) that $\mathcal{A}$ needs to fulfill is that for every x where $x \coloneqq x_0 = x_1$ and where the corresponding party has been corrupted, it needs to hold that $F_e(x, x_0) = F_e(x, x_1)$ for all the challenge queries (x_0, x_1), and all policy updates F_e. This matches exactly the validity requirements asked for $\mathcal{B}$ in the $\mathrm{AH}_\beta^{\mathsf{2\text{-}PHPE}}$ experiment, which is the requirement that $F_e(x, x_0) = F_e(x, x_1)$ for all encoding queries x, all encryption queries (x_0, x_1), and all key generation queries F_e. Therefore, it follows that the adversary $\mathcal{B}$ is a valid adversary with respect to the $\mathrm{AH}_\beta^{\mathsf{2\text{-}PHPE}}$ experiment and does not abort if the adversary $\mathcal{A}$ is a valid adversary in the game $\mathsf{G}_{2+\beta}$ $(\mathrm{AH}_\beta^{\mathsf{UPCS}})$.

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. □

4.3 Relationship between UPCS and 2-PHPE

Now, we show a simple compiler to realize 2-PHPE from *non-interactive* UPCS.[10] In particular, the predicate class supported by the resulting 2-PHPE scheme is the same that the underlying UPCS scheme supports. This establishes the formal equivalence of the primitives UPCS and 2-PHPE. This equivalence thus highlights the non-triviality in building any UPCS scheme from concrete assumptions. Let $\mathcal{P}_\lambda = \{P : \mathcal{X}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}\}$ be the class of two-input predicate UPCS schemes. We build a 2-PHPE scheme supporting the same class of predicates $\mathcal{P}_\lambda$ as follows:

[9]The check regarding the associated policies together with the correctness of the PHPE 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).

[10]For brevity, we will omit the term “non-interactive” and refer to the scheme as just UPCS in this section.

$\mathsf{2\text{-}PHPE.Setup}(1^\lambda)$: Let $P_0 \in \mathcal{P}_\lambda$ be any arbitrary predicate. (Without loss of generality, we can fix P_0 as the first lexicographically appearing policy in $\mathcal{P}_\lambda$.)

Initialize a UPCS authority $\mathfrak{A}$ with an input $(\textsc{Setup}, P_0)$ upon which it computes

$$(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{msk}_{\mathsf{UPCS}}, \mathsf{tok}_0) \leftarrow \mathsf{UPCS.Setup}(1^\lambda, P_0)$$

Output $(\mathsf{pp}, \mathsf{msk}) := (\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{msk}_{\mathsf{UPCS}})$.

$\mathsf{2\text{-}PHPE.KeyGen}(\mathsf{msk}, P)$: Parse $\mathsf{pp} = \mathsf{mpk}_{\mathsf{UPCS}}$, $\mathsf{msk} = \mathsf{msk}_{\mathsf{UPCS}}$ and the predicate P. Invoke the policy-update algorithm on input $(\textsc{Update}, P)$ to get a token associated to P as

$$\mathsf{tok}_P \leftarrow \mathsf{UPCS.PolUpd}(\mathsf{msk}_{\mathsf{UPCS}}, \mathsf{mpk}_{\mathsf{UPCS}}, P).$$

Output $\mathsf{sk}_P := \mathsf{tok}_P$.

$\mathsf{2\text{-}PHPE.Encode}(\mathsf{msk}, x_1)$: Parse $\mathsf{msk} = \mathsf{msk}_{\mathsf{UPCS}}$ and a set of attributes $x_1 \in \mathcal{X}_\lambda$. Initialize a party P_S by invoking $\mathfrak{A}$ on input $(\textsc{KeyGen}, P_S, x_1)$ to compute

$$(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{UPCS.KeyGen}(\mathsf{msk}_{\mathsf{UPCS}}, P_S, x_1).$$

Output $\mathsf{e}_{x_1} := (\mathsf{pk}_S, \mathsf{sk}_S, x_1)$.

$\mathsf{2\text{-}PHPE.Encrypt}(\mathsf{msk}, x_2)$: Parse $\mathsf{msk} = \mathsf{msk}_{\mathsf{UPCS}}$ and a set of attributes $x_2 \in \mathcal{X}_\lambda$. Initialize a party P_R by invoking $\mathfrak{A}$ on input $(\textsc{KeyGen}, P_R, x_2)$ to compute

$$(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{UPCS.KeyGen}(\mathsf{msk}_{\mathsf{UPCS}}, P_R, x_2).$$

Output $\mathsf{ct} := \mathsf{pk}_R$.

$\mathsf{2\text{-}PHPE.Decrypt}(\mathsf{sk}_P, (\mathsf{e}_{x_1}, \mathsf{ct}))$: Parse $\mathsf{pp} = \mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{sk}_P = \mathsf{tok}_P$ associated with predicate P, $\mathsf{e}_{x_1} = (\mathsf{pk}_S, \mathsf{sk}_S, x_1)$ associated with a party P_S, $\mathsf{ct} = \mathsf{pk}_R$ associated with a party P_R (with identifier R). Fix a message m. Run the signing algorithm on behalf of party P_S on input $(\textsc{Sign}, \mathsf{tok}_P, R, m)$ to compute $\sigma \leftarrow \mathsf{UPCS.Sign}(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{tok}_P, \mathsf{sk}^S, \mathsf{pk}^R, m)$. Run the verification algorithm on behalf of party P_R on input $(\textsc{Verify}, \mathsf{tok}_P, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ to compute and output the bit $b := \mathsf{UPCS.Verify}(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{tok}_P, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$.

Correctness follows directly from that of the UPCS scheme. Formally, consider any predicate $P \in \mathcal{P}_\lambda$ and inputs $x_1, x_2 \in \mathcal{X}_\lambda$, such that $P(x_1, x_2) = b$ for any $b \in \{0, 1\}$.

1. The $\mathsf{2\text{-}PHPE.Setup}$ initializes a UPCS authority with a fixed predicate P_0 (as described above) that outputs $(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{msk}_{\mathsf{UPCS}}, \mathsf{tok}_0) \leftarrow \mathsf{UPCS.Setup}(1^\lambda, P_0)$. Accordingly, the $\mathsf{2\text{-}PHPE.Setup}$ sets $(\mathsf{pp}, \mathsf{msk}) := (\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{msk}_{\mathsf{UPCS}})$.

2. For a key associated to predicate P, the $\mathsf{2\text{-}PHPE.KeyGen}$ procedure outputs a secret key $\mathsf{sk}_P = \mathsf{UPCS.tok}_P \leftarrow \mathsf{UPCS.PolUpd}(\mathsf{msk}_{\mathsf{UPCS}}, \mathsf{mpk}_{\mathsf{UPCS}}, P)$.

3. To encode input attributes x_1 for slot 1, the $\mathsf{2\text{-}PHPE.Encode}$ algorithm invokes a key generation for an initialized party P_S as $\mathsf{UPCS.KeyGen}(\mathsf{msk}_{\mathsf{UPCS}}, P_S, x_1)$ and sets $\mathsf{e}_{x_1} := (\mathsf{pk}_S, \mathsf{sk}_S, x_1)$.

4. To encrypt input attributes x_2 for slot 2, the $\mathsf{2\text{-}PHPE.Encrypt}$ algorithm again invokes a key generation for an initialized party P_R to compute $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{UPCS.KeyGen}(\mathsf{msk}_{\mathsf{UPCS}}, P_R, x_2)$ and sets $\mathsf{ct} := \mathsf{pk}^R$.

5. Observe that both, $\mathsf{e}_{x_1} = (\mathsf{pk}_S, \mathsf{sk}_S, x_1)$ and $\mathsf{ct} \coloneqq \mathsf{pk}^R$, correspond to UPCS keys held by parties initialized by the UPCS experiment. Therefore, the 2-PHPE.Decrypt algorithm simply signs a fixed message m with respect to the token tok_P and the keys $(\mathsf{sk}_S, \mathsf{pk}_R)$, and verifies it with respect to the same token and the key pair $(\mathsf{pk}_S, \mathsf{pk}_R)$.

Note that the predicate class supported by the UPCS scheme is inherited by the resulting 2-PHPE scheme. Further, since the 2-PHPE scheme initializes all the honest parties in the UPCS experiment, we have that $\mathsf{UPCS.Verify}(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{tok}_P, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma) = b = P(x_1, x_2)$ with probability 1. This concludes the formal correctness of the 2-PHPE scheme.

We next present the attribute-hiding proof.

Theorem 4.13. *Let* $\mathsf{UPCS} = (\mathsf{UPCS.Setup}, \mathsf{UPCS.PolUpd}, \mathsf{UPCS.KeyGen}, \mathsf{UPCS.Sign}, \mathsf{UPCS.Verify})$ *be an attribute-hiding non-interactive UPCS scheme supporting the two-input predicate class* $\mathcal{P}_\lambda = \{P : \mathcal{X}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}\}$*. Then the scheme* $\mathsf{2\text{-}PHPE} = (\mathsf{2\text{-}PHPE.Setup}, \mathsf{2\text{-}PHPE.KeyGen}, \mathsf{2\text{-}PHPE.Encode}, \mathsf{2\text{-}PHPE.Encrypt}, \mathsf{2\text{-}PHPE.Decrypt})$ *obtained above for the predicate class* $\mathcal{P}_\lambda$ *is also attribute-hiding.*

Proof. Assume $\mathcal{A}$ to be a valid PPT adversary against the 2-PHPE scheme in the $\mathrm{AH}_\beta^{\mathsf{2\text{-}PHPE}}$ experiment as per Definition 4.2. We then construct an adversary $\mathcal{A}'$ that internally uses $\mathcal{A}$ as a subroutine and plays an execution in the UPCS environment to break attribute-hiding of UPCS as follows:

- $\mathcal{A}'$ triggers the UPCS authority $\mathfrak{A}$ on input $(\textsc{Setup}, P_0)$ to get $(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{tok}_0)$ and feeds $\mathsf{pp} \coloneqq \mathsf{mpk}_{\mathsf{UPCS}}$ to $\mathcal{A}$.
- For a key query by $\mathcal{A}$ for some predicate $P \in \mathcal{P}_\lambda$, $\mathcal{A}'$ invokes $\mathfrak{A}$ on input $(\textsc{Update}, P)$ and gets a token tok_P for P. Upon receiving it, $\mathcal{A}'$ feeds $\mathsf{sk}_P \coloneqq \mathsf{tok}_P$ to $\mathcal{A}$.
- When $\mathcal{A}$ queries to *encode* an attribute $x_{1,i} \in \mathcal{X}_\lambda$, $\mathcal{A}'$ initializes a party P_S^i which invokes $\mathfrak{A}$ on input $(\textsc{Test-KeyGen}, P_S^i, (x'_{i,0}, x'_{i,1}))$ setting $x'_{i,0} = x'_{i,1} = x_{1,i}$. After P_S^i receives $(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{pk}_S^i, \mathsf{sk}_S^i, x_{1,i})$ and outputs $(\textsc{Initialized}, \mathsf{pk}_S^i)$, $\mathcal{A}'$ obtains pk_S^i from this output. Next, $\mathcal{A}'$ inputs $\textsc{Corrupt}$ to party P_S^i and further obtains $(\mathsf{sk}_S^i, x_{1,i})$. $\mathcal{A}'$ then feeds $\mathsf{e}_{x_{1,i}} = (\mathsf{pk}_S^i, \mathsf{sk}_S^i, x_{1,i})$ to $\mathcal{A}$.
- When $\mathcal{A}$ issues a challenge query $(x_{2,j}^0, x_{2,j}^1) \in (\mathcal{X}_\lambda)^2$, $\mathcal{A}'$ initializes a party P_R^j which invokes $\mathfrak{A}$ on input $(\textsc{Test-KeyGen}, P_R^j, (x_{2,j}^0, x_{2,j}^1))$. After P_R^j receives $(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{pk}_R^j, \mathsf{sk}_R^j, x_{2,j}^b)$ and outputs $(\textsc{Initialized}, \mathsf{pk}_R^j)$, $\mathcal{A}'$ obtains pk_R^j from this output and feeds $\mathsf{ct} \coloneqq \mathsf{pk}_R^j$ to $\mathcal{A}$.
- When $\mathcal{A}$ returns a bit b', $\mathcal{A}'$ outputs the same.

It is easy to observe that the UPCS execution enviroment lets $\mathcal{A}'$ simulate a distribution identical to the $\mathrm{AH}_\beta^{\mathsf{2\text{-}PHPE}}$ experiment in the view of its internal 2-PHPE adversary $\mathcal{A}$. The output distribution of $\mathcal{A}'$ is also exactly the same as that of $\mathcal{A}$. Hence, we have $\mathsf{Adv}_{\mathsf{2\text{-}PHPE},\mathcal{A}}^{\mathrm{AH}} = \mathsf{Adv}_{\mathsf{UPCS},\mathcal{A}'}^{\mathrm{AH}}$. It remains to establish that $\mathcal{A}'$ is a valid UPCS adversary, given $\mathcal{A}$ is valid. For this, we first observe the following:

1. Any key generation query for a predicate P put forth by $\mathcal{A}$ is answered with a token tok_P for the same predicate P by the UPCS authority $\mathfrak{A}$.
2. Any encoding query issued by $\mathcal{A}$ for attributes $x_{1,i}$ in slot 1 is essentially translated to a corruption query by $\mathcal{A}'$. This is already fine since this encoding is not supposed to hide $x_{1,i}$. Hence, as per the UPCS attribute-hiding experiment, $\mathcal{A}'$ initializes the party P_S^i which further inputs $(\textsc{Test-KeyGen}, P_{S,}^i, (x'_{i,0}, x'_{i,1}))$ setting $x'_{i,0} = x'_{i,1} = x_{1,i}$ to $\mathfrak{A}$ and outputs $(\textsc{Initialized}, \mathsf{pk}_S^i)$ once it gets $(\mathsf{mpk}_{\mathsf{UPCS}}, \mathsf{pk}_S^i, \mathsf{sk}_S^i, x_{1,i})$ from $\mathfrak{A}$. Next, $\mathcal{A}'$ further inputs $\textsc{Corrupt}$ to P_S^i to obtain $(\mathsf{sk}_S^i, x_{1,i})$ and feeds $\mathsf{e}_{x_{1,i}}$ accordingly to $\mathcal{A}$.

3. Any encryption query issued by $\mathcal{A}$ for a pair of attributes $(x^0_{2,j}, x^1_{2,j})$ in slot 2 is again answered by $\mathcal{A}'$ in a similar manner as above by freshly initializing a party P^j_R (except that now it is not corrupted). In particular, $\mathcal{A}'$ gets to see pk^j_R once P^j_R is initialized and accordingly feeds $\mathsf{ct} := \mathsf{pk}^j_R$ to $\mathcal{A}$.

4. Note that $\mathcal{A}$ can issue all the above three queries adaptively, while maintaining its validity criteria that for all i, j and all key queries P, it must hold that $P(x_{1,i}, x^0_{2,j}) = P(x_{1,i}, x^1_{2,j})$.

Clearly, the validity of $\mathcal{A}$ above implies $P(x'_{i,0}, x^0_{2,j}) = P(x_{1,i}, x^0_{2,j}) = P(x_{1,i}, x^1_{2,j}) = P(x'_{i,1}, x^1_{2,j})$ for all predicates P and for all the query indices i, j. Besides, note that $\mathcal{A}'$ never needs to run an instruction of the type $(\textsc{Sign}, \mathsf{tok}_P, R, m)$ in this environment. Thus, all other validity criteria for $\mathcal{A}'$ related to tokens and signing oracles are satisfied vacously. This concludes the proof. $\square$

5 Interactive Updatable Policy-Compliant Signatures

In this section, we present our interactive updatable policy-compliant signature schemes.

The first scheme relies on signatures, non-interactive zero-knowledge proofs and two-party computation. This scheme requires interaction between the sender and receiver in *each* update for a signature generation. Our second scheme instead requires only a *single* interaction between the sender and receiver that allows a signature generation for all updates. To facilitate this, we need to additionally rely on a predicate encryption scheme. We present both these schemes formally in Sections 5.1 and 5.2 below.

5.1 Interactive UPCS using Two-Party Computation

We start by presenting the first scheme that relies only on two-party computation. Here, the public key of a party simply consists of a verification key of a signature scheme vk and its secret key contains the corresponding signing key sk as well as its attribute set x. Additionally, the authority binds these keys together by generating signatures for both of them involving the verification key vk, i.e., it generates a signature over vk for the public key and a signature over (vk, x) for the secret key. Looking ahead, they are used inside the two-party computation protocol in the signature generation to allow for the policy computation. The authority generates a policy update for a policy F' by simply signing it and outputting the corresponding signature σ_{tok} together with the policy.

```
$\underline{\mathsf{Setup}(1^\lambda, F_{\mathsf{init}})}$:
$\mathsf{CRS}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^\lambda)$
$\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, F_{\mathsf{init}})$
$\mathsf{mpk} := (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$
$\mathsf{msk} := (\mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{tok}})$
$\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \sigma_{\mathsf{tok}})$
Return $(\mathsf{mpk}, \mathsf{msk}), \mathsf{tok}_{\mathsf{init}}$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, P_i, x)}$:
Parse $\mathsf{msk} = (\mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{tok}})$
$(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}(1^\lambda)$
$\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, P_i))$
$\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, x))$
$\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, P_i, \sigma_{\mathsf{pub}}), \mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, x, \sigma_{\mathsf{priv}})$
Return $(\mathsf{pk}, \mathsf{sk})$
```

```
$\underline{\mathsf{PolUpd}(\mathsf{mpk}, \mathsf{msk}, F_{\mathsf{upd}})}$
Parse $\mathsf{mpk} = (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$
      $\mathsf{msk} = (\mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{tok}})$
$\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, F_{\mathsf{upd}})$
$\mathsf{tok}_{\mathsf{upd}} := (F_{\mathsf{upd}}, \sigma_{\mathsf{tok}})$
Return $\mathsf{tok}_{\mathsf{upd}}$

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

Figure 10: The setup, policy update, key generation and verification procedures of our interactive UPCS scheme.

For the signature generation, the sender and the receiver interact in a two-party computation protocol where the sender inputs its secret key $\mathsf{sk}_S := (\mathsf{vk}_S, x_S)$ and the public key of the receiver pk_R and the receiver inputs its secret key $\mathsf{sk}_R := (\mathsf{vk}_R, x_R)$ as well as the public key of the sender pk_S. Furthermore, both of the parties input the current token $(F', \sigma_{\mathsf{tok}})$. The circuit that the two-party computation protocol computes first verifies that the secret keys input by the parties correspond to the public keys, i.e., that sk_S corresponds to pk_S and sk_R corresponds to pk_R, by checking the contained information as well as verifying the signatures. Furthermore, it also verifies the signature of the token σ_{tok}. Afterwards, it evaluates the policy, i.e., checks that $F'(x_S, x_R) = 1$, and, if the check succeeds, it generates a non-interactive zero-knowledge proof π that the sender and receiver together fulfill the policy. In this proof, the verification keys of the sender and the receiver vk_S and vk_R are part of the statement. After the execution of the two-party computation protocol, the sender obtains π and uses it together with the message m and its signing key to generate the signature σ over the tuple (m, pk_R, π). The sender then outputs $((m, \mathsf{pk}_R, \pi), \sigma)$ as the final signature.

To verify a signature, the signatures associated with the public keys of the sender and the receiver are verified, as well as the signature of the token for which it has been generated. If these verifications succeed, then the information inside the public keys and the token can be used to verify the proof π of the signature. In a last step, the signature σ generated over the proof and the message (m, pk_R, π) is verified using the verification key vk_S of the sender. If all of these verification checks succeed, then the signature is deemed valid.

If the sender wants to generate another signature for the same receiver under the same token, it can simply reuse the proof π that has been output by the two-party computation protocol and no additional interaction is required. We describe the formal scheme in Figures 10 and 11.

```
$\Pi_{\text{sign}}(\mathsf{mpk}, \mathsf{tok}, \mathsf{sk}_S, \mathsf{pk}_S, R, m)$ for signer $S$:
Parse $\mathsf{mpk} = (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$
        $\mathsf{tok} = (F, \sigma_{\mathsf{tok}})$
        $\mathsf{sk}_S = (\mathsf{vk}_{\mathsf{P}}^S, \mathsf{sk}_{\mathsf{P}}^S, x_S, \sigma_{\mathsf{priv}}^S)$
        $\mathsf{pk}_S = (\mathsf{vk}_{\mathsf{P}}^S, P_S, \sigma_{\mathsf{pub}}^S)$
Obtain $\mathsf{pk}_R = (\mathsf{vk}_{\mathsf{P}}^R, P_R, \sigma_{\mathsf{pub}}^R)$ from $R$ (cf. Sec. 3).
If this algorithm has already been executed between $S$ and $R$ for this token $\mathsf{tok}$,
proceed to "If $\pi \neq \bot, \ldots$" Otherwise, proceed as follows:
If $\mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{tok}}, F, \sigma_{\mathsf{tok}}) = 0$, return $\bot$.
The remainder of this algorithm is an interactive process between $S$ and $R$:
$S$: If $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}^R, P_R), \sigma_{\mathsf{pub}}^R) = 0$,
    return $\bot$.
$R$: If $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}^S, P_S), \sigma_{\mathsf{pub}}^S) = 0$,
    return $\bot$
$S$ and $R$ execute $\Pi$ that computes the circuit described in Figure 12.
After the execution of $\Pi$, $S$ obtains $\pi$.
If $\pi \neq \bot$, $S$ computes $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{P}}^S, (m, \mathsf{pk}_R, \pi))$,
sets $\sigma := (\pi, \sigma')$ and returns $(m, \mathsf{pk}_R, \sigma)$.
```

Figure 11: Our interactive 2PC-based UPCS scheme. It uses a NIZK proof system NIZK, a 2PC protocol Π in the CRS model, and four digital signature schemes $\mathsf{DS}_{\mathsf{pub}}, \mathsf{DS}_{\mathsf{priv}}, \mathsf{DS}_{\mathsf{tok}}$ and DS_{P}.

5.1.1 Correctness and Security

The correctness of the scheme follows directly from the correctness of the underlying schemes: DS, NIZK and that of $\mathsf{2PC}$. For security, unforgeability follows from unforgeability of the signature schemes and the (knowledge) soundness of the NIZK proof. Further, the attribute-hiding of the UPCS scheme follows from the simulatability of $\mathsf{2PC}$ and the zero-knowledge property of the NIZK. We present the formal proofs below. We note in passing that when switching to a model where we cannot erase the randomness during proof generation, we obtain an analogous result when switching to a NIZK that supports adaptive corruptions [GOS06].

Unforgeability

Now, we proceed to the unforgeability proof of our scheme.

Theorem 5.1. *Let* $\mathsf{DS}_{\mathsf{pub}} = (\mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify})$, $\mathsf{DS}_{\mathsf{tok}} = (\mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify})$, $\mathsf{DS}_{\mathsf{priv}} = (\mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify})$ *and* $\mathsf{DS}_{\mathsf{P}} = (\mathsf{DS}_{\mathsf{P}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{P}}.\mathsf{Verify})$ *be EUF-CMA secure signature schemes, and let* $\mathsf{NIZK} = (\mathsf{NIZK}.\mathsf{Setup}, \mathsf{NIZK}.\mathsf{Prove}, \mathsf{NIZK}.\mathsf{Verify})$ *be an extractable proof system and* Π *be an adaptively UC-secure two-party computation protocol for the specified circuit in Figure 12, then the construction* $\mathsf{UPCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figures 10 and 11, is existentially unforgeable.*

Proof. Using the composition theorem, we can effectively work in a world where parties use $\mathcal{F}_C$ to evaluate a circuit C and rely on the correctness of the computation to ensure the computed values

Circuit $C_{\mathsf{CRS}_{\mathsf{NIZK}},\mathsf{vk}_{\mathsf{tok}},\mathsf{vk}_{\mathsf{priv}}}$:

Inputs: S uses $(\mathsf{pk}_R := (\mathsf{vk}^R_{\mathsf{P},S}, P_R, \sigma^R_{\mathsf{pub},S}), \mathsf{sk}_S := (\mathsf{vk}^S_{\mathsf{P},S}, \cdot, x_S, \sigma^S_{\mathsf{priv}}), \mathsf{tok}_S := (F_S, \sigma_{\mathsf{tok},S}))$ as its input and R uses $(\mathsf{pk}_S := (\mathsf{vk}^S_{\mathsf{P},R}, P_S, \sigma^S_{\mathsf{pub},R}), \mathsf{sk}_R := (\mathsf{vk}^R_{\mathsf{P},R}, \cdot, x_R, \sigma^R_{\mathsf{priv}}),$
$\mathsf{tok}_R := (F_R, \sigma_{\mathsf{tok},R}))$ as its input.

Check that:

- $F := F_S = F_R, \sigma_{\mathsf{tok}} := \sigma_{\mathsf{tok},S} = \sigma_{\mathsf{tok},R}$ and that $\mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{tok}}, (F, \sigma_{\mathsf{tok}})) = 1$.
- $\mathsf{vk}^R_{\mathsf{P}} := \mathsf{vk}^R_{\mathsf{P},S} = \mathsf{vk}^R_{\mathsf{P},R}, \sigma^R_{\mathsf{pub},S} = \sigma^R_{\mathsf{pub},R}, \mathsf{vk}^S_{\mathsf{P}} := \mathsf{vk}^S_{\mathsf{P},R} = \mathsf{vk}^S_{\mathsf{P},S}, \sigma^S_{\mathsf{pub},R} = \sigma^S_{\mathsf{pub},S}$.
- $\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}^S_{\mathsf{P},S}, x_S), \sigma^S_{\mathsf{priv}}) = 1$ and
  $\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}^R_{\mathsf{P},R}, x_R), \sigma^R_{\mathsf{priv}}) = 1$.
- $F(x_S, x_R) = 1$.

If one of these checks fails output $\perp$. If all of these checks are successful, compute:

- $\pi \leftarrow \mathsf{NIZK}.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{NIZK}}, ((F, \sigma_{\mathsf{tok}}), \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P}}, \mathsf{vk}^R_{\mathsf{P}}), (x_S, x_R, \sigma^S_{\mathsf{priv}}, \sigma^R_{\mathsf{priv}}))$
  for the relation described in Figure 13.

Output: π to S.

Figure 12: The circuit that is being computed by the 2PC of the interactive UPCS scheme in Figure 11.

Relation R_{ZK}:

Instance: $x = (\mathsf{tok} := (F, \sigma_{\mathsf{tok}}), \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P}}, \mathsf{vk}^R_{\mathsf{P}})$

Witness: $w = (x_S, x_R, \sigma^S_{\mathsf{priv}}, \sigma^R_{\mathsf{priv}})$

$R_{\mathrm{ZK}}(x, w) = 1$ if and only if:

$\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}^P_{\mathsf{P}}, x_P), \sigma^P_{\mathsf{priv}}) = 1$, for $P \in \{S, R\}$ and $F(x_S, x_R) = 1$

Figure 13: Relation used for the NIZK inside the 2PC of the interactive UPCS scheme in Figure 12.

are as specified. The proof of this theorem proceeds very similar to the proof of the non-interactive scheme. We define the same events as in the proof of Theorem 4.3 and recap them here:

- Event $\mathsf{KeyForge}_{\mathcal{A}}$: The adversary $\mathcal{A}$ terminates with output $(\mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ where the public key pk_S (resp. pk_R) does not belong to any initialized party P_S (resp. P_R) (by means of an invocation $(\textsc{KeyGen}, P_S, x_S)$, (resp. $(\textsc{KeyGen}, P_R, x_R)$)).

- Event $\mathsf{KeyColl}_{\mathcal{A}}$: The adversary terminates and it holds that there are two parties P_i and P_j where $i \neq j$ such that for the corresponding public keys $\mathsf{pk}_i = (\mathsf{vk}_i, \cdot, \cdot, \cdot)$ and $\mathsf{pk}_j = (\mathsf{vk}_j, \cdot, \cdot, \cdot)$ it holds that $\mathsf{vk}_i = \mathsf{vk}_j$.

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

- Event $\mathsf{WIN1}_{\mathcal{A}}$: The adversary generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where the public key pk is associated with some initialized party P_S that is not corrupted and was never invoked on input $(\textsc{Sign}, P_S, \mathsf{tok}, \mathsf{pk}_R, m)$.

- Event $\mathsf{WIN2}_{\mathcal{A}}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where the update token tok^* is not associated with a policy F^* that has never been an input to the authority $(\textsc{Update}, F^*)$. We also denote this event as $\mathsf{PolicyForge}_{\mathcal{A}}$.

- Event $\mathsf{WIN3}_{\mathcal{A}}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where there is a policy F' such that a query $(\textsc{Update}, F')$ has been asked, and there are attributes x_S and x_R associated to some initialized parties P_S and P_R with public keys pk_S and pk_R, respectively (as the result of inputs $(\textsc{KeyGen}, P_S, x_S)$ and $(\textsc{KeyGen}, P_R, x_R)$ to $\mathfrak{A}$), such that $F'(x_S, x_R) = 0$.

The bounds for events $\mathsf{KeyForge}_{\mathcal{A}}$ and $\mathsf{KeyColl}_{\mathcal{A}}$ follow as in the non-interactive scheme, i.e., by Lemmata 4.4 and 4.5:

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

for adversaries $\mathcal{B}_1$ and $\mathcal{B}'_2$ which are constructed based on $\mathcal{A}$ and have roughly the same efficiency as $\mathcal{A}$ and where q are the number of key generation queries $\textsc{KeyGen}$.

Also the bounds of the events $\mathsf{WIN1}_{\mathcal{A}}, \mathsf{WIN2}_{\mathcal{A}}$ and $\mathsf{WIN3}_{\mathcal{A}}$ are the same as in the non-interactive case.

$$\begin{aligned}
&\Pr[\mathsf{WIN1}_{\mathcal{A}}] \quad \leq q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{P}},\mathcal{B}''_2}(\lambda),\\
&\Pr[\mathsf{WIN2}_{\mathcal{A}}/\ \mathsf{PolicyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{tok}},\mathcal{B}'_1}(\lambda) \text{ and}\\
&\Pr[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]\\
&\qquad\qquad \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK},\mathcal{B}'},
\end{aligned}$$

where the adversaries $\mathcal{B}'_1, \mathcal{B}''_2$, $\mathcal{B}_3$, $\mathcal{B}_4$, and distinguisher $\mathcal{B}'$ are constructed based on $\mathcal{A}$ and have roughly the same efficiency as $\mathcal{A}$.

The proof of the bounds for the events $\mathsf{WIN1}_{\mathcal{A}}$ and $\mathsf{WIN2}_{\mathcal{A}}$ proceed in almost exactly the same way as described in the proofs of Lemmata 4.6 and 4.7. The proof of the bound for $\mathsf{WIN3}_{\mathcal{A}}$ differs slightly from the proof in the non-interactive case, therefore, we recap the adjusted proof in Lemma 5.2.

By definition of the events, we have

$$\begin{aligned}
\Pr[\mathsf{WIN}_{\mathcal{A}}] \leq\, &\Pr[\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}]\\
&+ \Pr[\mathsf{WIN}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]\\
\leq\, &\Pr[\mathsf{KeyColl}_{\mathcal{A}}] + \Pr[\mathsf{KeyForge}_{\mathcal{A}}] + \Pr[\mathsf{PolicyForge}_{\mathcal{A}}]\\
&+ \Pr[\mathsf{WIN1}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]\\
&+ \underbrace{\Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]}_{=0,\text{ since } \mathsf{WIN2}_{\mathcal{A}}=\mathsf{PolicyForge}_{\mathcal{A}}}\\
&+ \Pr[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}].
\end{aligned}$$

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$^{\mathsf{DS}_{\mathsf{P}}}$. Therefore, the theorem follows. ☐

Lemma 5.2. *Consider the unforgeability experiment and let* $\mathsf{WIN3}_{\mathcal{A}}$ *be defined as above. We can construct adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *and a distinguisher* $\mathcal{B}'$ *such that*

$$\begin{aligned}
&\Pr[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]\\
&\qquad\qquad\qquad \leq \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda).
\end{aligned}$$

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

$$\Pr[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]$$

$$\leq \Pr_{\mathrm{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right] + \mathsf{Adv}_{\mathsf{NIZK},\mathcal{B}'}^{\mathrm{CRS}},$$

where $\Pr_{\mathrm{Hyb}}[.]$ makes explicit that this probability is taken w.r.t. experiment EUF-CMA$_{\mathrm{Hyb}}^{\mathsf{UPCS}}$. 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 $\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}$ occurs (which is computable by $\mathcal{B}'$ that manages all key-sets).

We build an adversary $\mathcal{B}_1$ that simulates EUF-CMA$_{\mathrm{Hyb}}^{\mathsf{UPCS}}$ towards $\mathcal{A}$ when interacting with the underlying EUF-CMA$^{\mathsf{DS}_{\mathsf{priv}}}$ experiment. We show that if $\mathcal{A}$ outputs $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ as defined in event $\mathsf{WIN2}$, it can be used as a forgeability attack in the EUF-CMA$^{\mathsf{DS}_{\mathsf{priv}}}$ experiment, unless a certain failure event $\mathsf{Fail}_{\mathrm{ext}}$ 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 EUF-CMA$^{\mathsf{priv}}$ experiment and the policy F_{init} from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}_1$ generates $\mathsf{CRS}_{\mathsf{NIZK}} \leftarrow E_1(1^\lambda)$, $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^\lambda)$ and sets $\mathsf{mpk} := (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$. Furthermore, $\mathcal{B}$ generates $\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}(\mathsf{sk}_{\mathsf{tok}}, F_{\mathsf{init}})$. Afterwards, it sets $\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \sigma_{\mathsf{tok}})$ and sends mpk and $\mathsf{tok}_{\mathsf{init}}$ to the adversary $\mathcal{A}$. The adversary $\mathcal{B}$ also initializes the counter $e = 2$.

If the adversary $\mathcal{A}$ asks a policy update query $(\textsc{Update}, F')$, the adversary $\mathcal{B}$ generates $\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}(\mathsf{sk}_{\mathsf{tok}}, F')$ and outputs $\mathsf{tok}' := (F', \sigma'_{\mathsf{tok}})$ to $\mathcal{A}$. Afterwards, it increases e, i.e., $e := e + 1$.

For a key generation query $(\textsc{KeyGen}, P_i, x)$, $\mathcal{B}_1$ samples a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$, generates the signature $\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, P_i))$ and submits $(\mathsf{vk}_{\mathsf{P}}, x)$ to the sign oracle of the underlying EUF-CMA$^{\mathsf{DS}_{\mathsf{priv}}}$ experiment to obtain the signature σ_{priv}. Then, $\mathcal{B}_1$ sets $\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, P_i, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, x, \sigma_{\mathsf{priv}})$ 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 $(\textsc{Corrupt}, P_j)$, the adversary $\mathcal{B}_1$ searches for the key sk that is associated with P_j. If no such entry exists, the adversary $\mathcal{B}_1$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk to $\mathcal{A}$.

If the adversary $\mathcal{A}$ asks a signing query $(\textsc{Sign}, \mathsf{tok}, P_j, \mathsf{pk}_R := (\mathsf{vk}_R, P_R, \sigma_{\mathsf{pub}}^R), m)$, then the adversary $\mathcal{B}_1$ checks that a key has been generated for P_j and that pk_R has been the reply to a previous key generation query $\textsc{KeyGen}$. If this is not the case, the adversary $\mathcal{B}_1$ aborts. Otherwise, it continues with the execution of Sign, using the key sk of party P_j and terminates once Sign terminates.

When $\mathcal{A}$ terminates with $(\mathsf{tok}^* := (F^*, \sigma_{\mathsf{tok}}^*), \mathsf{pk}_S^* := (\mathsf{vk}_{\mathsf{P},S}^*, P_S^*, \sigma_{\mathsf{pub},S}^*), \mathsf{pk}_R^* := (\mathsf{vk}_{\mathsf{P},R}^*, P_R^*, \sigma_{\mathsf{pub},R}^*),$ $m^*, \sigma^* := (\pi^*, \sigma'))$ $\mathcal{B}_1$ first checks whether the conditions of event $\mathsf{KeyForge}_{\mathcal{A}}$ hold, if this is the case, it aborts. It also verifies that the conditions of event $\mathsf{WIN3}_{\mathcal{A}}$ hold and in case this is true, it first calls $(x_S^*, x_R^*, \sigma_{\mathsf{priv},S}^*, \sigma_{\mathsf{priv},R}^*) \leftarrow E_2(\mathsf{CRS}, (\mathsf{tok}^*, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{P},S}^*, \mathsf{vk}_{\mathsf{P},R}^*), \pi^*)$ and checks whether $(x := (\mathsf{tok}^*, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{P},S}^*, \mathsf{vk}_{\mathsf{P},R}^*), w := (x_S^*, x_R^*, \sigma_{\mathsf{priv},S}^*)) \in R_{\mathrm{ZK}}$ (which is efficiently checkable) and if this is the case, it submits either $((\mathsf{vk}_{\mathsf{P},S}^*, x_S^*), \sigma_{\mathsf{priv},S}^*)$ or $((\mathsf{vk}_{\mathsf{P},R}^*, x_R^*), \sigma_{\mathsf{priv},R}^*)$, depending on which one has not been previously queried to the underlying challenger, as a forgery to the underlying experiment EUF-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 events in this emulation are defined as in the experiment $\text{EUF-CMA}^{\mathsf{UPCS}}_{\text{Hyb}}$ 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 $\mathcal{A}$ have previously been the answer to a KeyGen query. Therefore, there are parties P_j and P_k such that pk^*_S is the key of P_j and pk^*_R is the key of P_k and where the keys contain vk_i and are associated with x_i for $i \in \{j, k\}$. Let us fix these two indices. Furthermore, we can assume that $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{tok}^*, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_j, \mathsf{vk}_k), \pi^*) = 1$ as otherwise, $\mathsf{WIN3}$ does not hold. Additionally, we know that $F^*(x_j, x_k) = 0$ and since all keys are generated honestly by $\mathcal{B}$, for the keys of party P_j $(\mathsf{pk}_j, \mathsf{sk}_j = (\cdot, \cdot, x_j, \sigma_{\mathsf{priv},j}))$, it holds that $x_j \neq x^*_S$. Since by uniqueness, $\mathcal{B}_1$ has only submitted the query (vk_j, x_j) to the signing oracle of $\text{EUF-CMA}^{\mathsf{DS}_{\mathsf{priv}}}$, the pair $((\mathsf{vk}^*_{\mathsf{P},S} = \mathsf{vk}_j, x^*_S \neq x_j), \sigma^*_{\mathsf{priv},S})$ 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

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1} = \Pr_{\text{Hyb}}[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\text{ext}}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}].$$

This results in

$$\begin{aligned}\Pr_{\text{Hyb}}&\left[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right] \\ &= \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_1} + \underbrace{\Pr_{\text{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \mathsf{Fail}_{\text{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right]}_{\leq \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}}.\end{aligned}$$

It is straightforward to obtain an adversary $\mathcal{B}_2$ (based on $\mathcal{A}$) which has an advantage $\mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_2} = \Pr_{\text{Hyb}}\left[\mathsf{WIN3}_{\mathcal{A}} \cap \mathsf{Fail}_{\text{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right]$. 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 $(\mathsf{tok}^* := (F^*, \sigma^*_{\mathsf{tok}}), \mathsf{pk}^*_S := (\mathsf{vk}^*_{\mathsf{P},S}, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_{\mathsf{P},R}, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}_2$ behaves as $\mathcal{B}_1$ but does not execute the final steps running the extractor, but instead just outputs $(x := ((F^*, \sigma^*_{\mathsf{tok}}), \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{vk}^*_{\mathsf{P},R}), \pi^*)$ in case the conditions of $\mathsf{WIN3}$ are satisfied (note that the extractor is run as part of the experiment in Definition 2.6). As above, the emulation towards $\mathcal{A}$ is perfect until the point where $\mathcal{B}_2$ 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 $\mathsf{WIN3}$) but the extraction produces a witness w such that $(x, w) \notin R_{\text{ZK}}$. Therefore, the statement follows. □

Attribute-Hiding

After proving the unforgeability, we prove the attribute-hiding of our scheme.

Theorem 5.3. *Let Π be an adaptively UC-secure two-party computation protocol for the specified circuit in Figure 12, let $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ be a NIZK proof system (for the relation R_{ZK} of Figure 9) and let $\mathsf{DS}_{\mathsf{pub}} = (\mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify})$ and $\mathsf{DS}_{\mathsf{tok}} = (\mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify})$ be an unforgeable signature scheme, then the construction $\mathsf{UPCS} = (\mathsf{Setup}, \mathsf{PolUpd}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, defined in Figures 10 and 11, is attribute hiding.*

Proof. The proof of this theorem proceeds very similar to the proof of the non-interactive UPCS scheme with the difference that, instead of relying on the attribute-hiding property of the two-input partially-hiding predicate encryption scheme, we need to rely on the security of the two-party computation protocol.

To prove this statement, we use a hybrid argument with the following hybrids:

Hybrid H_0: This hybrid is defined as the attribute-hiding game for the case that $b = 0$.

Hybrid H_1: In this hybrid, we answer the SIGN query asked by the adversary differently. We denote the modified sign query as SIGN′. A SIGN′ query is defined as a SIGN query with the difference that it is only answered if the queried receiver key has been previously output as an answer to a TEST-KEYGEN query and the token has been previously output as an answer to a UPDATE query, i.e., for a query $(\textsc{Sign}, \mathsf{tok}_e, \mathsf{pk}_i, m)$ to P_j the public pk_i has been the answer to a query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ and the token tok_e has been the answer to a query $(\textsc{Update}, F_e)$. If this is not the case then the reply ot the SIGN′ query is $\perp$. The indistinguishability of H_0 and H_1 is justified by the same reasoning as we have seen for the non-interactive scheme (Figure 8). Namely, by relying on the Lemmata 4.4 and 4.7, as in the proof of Theorem 5.1. We prove this more formally in Lemma 5.4.

Hybrid H_2: In this hybrid, we change the way the output of every SIGN′ query that involves party P_j, which has obtained its key using a TEST-KEYGEN query, is computed. To compute the answer to the SIGN′ query we replace the execution of the protocol Π with an ideal execution of functionality $\mathcal{F}_C$ by relying on the composition theorem. Intuitively, what is happening is that in case that P_j is acting as the sender in the signing query, i.e., it is acting as the party P_1 in the underlying two-party computation protocol, simulated messages (that do not depend on the secret values of the honest party) are generated for P_i, the receiver of the transaction (corresponding to P_2 of the two-party computation protocol). This happens without any observable difference by the security of the 2PC. The case that the party P_j is acting as the receiver is analogous. The transition from H_1 to H_2 is stated as Lemma 5.5 for completeness.

Hybrid H_3: In this hybrid, we switch from an honestly generated $\mathsf{CRS}_{\mathsf{NIZK}}$ and honestly generated proofs in the singing queries to a simulated $\mathsf{CRS}_{\mathsf{NIZK}}$ and simulated proofs in the signing queries. That is, upon a PCS signing query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ to P_j, we find the attributes that have been used to generate the key pk_i for P_i, i.e., we find the query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ that was answered using pk_i, as well as the policy F_e that corresponds to the token tok_e, i.e., the query $(\textsc{Update}, F_e)$ which has been answered using tok_e. Further, we also obtain the attributes that are associated with party P_j, i.e., we search for the corresponding query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$. In the next step, we check that $F_e(x_{j,0}, x_{i,0}) = 1$. If this is the case, then we simulate the proof π that is generated inside the two-party computation protocol using the NIZK simulator on input the trapdoor and $(\mathsf{tok}_e, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{P}}^S, \mathsf{vk}_{\mathsf{P}}^R)$ where $\mathsf{vk}_{\mathsf{P}}^S$ and $\mathsf{vk}_{\mathsf{P}}^R$ are part of the public keys of the sender (P_i or P_j) and receiver (P_j or P_i). This simulated proof is then used as the output of the 2PC protocol. In any other case (in particular where associated attributes do not satisfy the policy), we output $\perp$. For more details, we refer to the proof of Lemma 5.6, where we argue the indistinguishability of H_2 and H_3 by relying on the zero-knowledge property of NIZK.

Hybrid H_4: In this hybrid, we answer the TEST-KEYGEN queries using the attributes x_1 instead of x_0. This transition is possible since, at this point, the generated signatures are completely independent of the attributes associated with the keys and, furthermore, a valid adversary can only ask corruption queries for a key where $x_0 = x_1$. Taking these two facts into account directly results in the indistinguishability between the hybrids H_3 and H_4.

Hybrid H_5: In this hybrid, we change back from a simulated CRS and simulated proofs π in the signing queries to an honestly generated CRS and honestly generated proofs π. In more detail, we use as a witness for the generation of the proof all the information that is part of the secret or public key, depending on if the party is acting as a sender or as a receiver in the query. In this hybrid, the attributes used as the witness now correspond to x_1 instead of x_0. Symmetrical to the transition from H_2 to H_3, this transition can be argued using the zero-knowledge property of NIZK, we refer to the proof of Lemma 5.6 for further details.

Hybrid H_6: In this hybrid, we change from a simulated execution of the two-party protocol to an honest execution of the protocol. As in the previous hybrid, the private information that the involved parties use in this hybrid are now w.r.t. x_1. Symmetrical to the transition from H_1 to H_2, this transition can be argued using the security of the two-party computation protocol Π. We refer to the proof of Lemma 5.5 for further details.

Hybrid H_7: This hybrid corresponds to the attribute-hiding game using $b = 1$ as its input. In this game, we change the behavior of the signing queries back from $\textsc{Sign}'$ to $\textsc{Sign}$. Symmetrical to the transition from H_0 to H_1, this transition can be argued using the unforgeability of public keys. We refer to the proof of Lemma 5.4 for further details.

From the definition of the hybrids it is clear that H_0 corresponds to the attribute-hiding game with $b = 0$ and H_7 corresponds to the attribute-hiding game with $b = 1$. Since it holds that $\mathsf{H}_0 \approx_c \cdots \approx_c \mathsf{H}_7$, the theorem follows. $\square$

Lemma 5.4 (Indistinguishability of H_0 to H_1)**.** *Let $\mathsf{DS}_{\mathsf{pub}}$ and $\mathsf{DS}_{\mathsf{tok}}$ be existentially-unforgeable signature scheme, then the hybrids H_0 and H_1 are computationally indistinguishable.*

Proof (Sketch). As described above, the difference between the hybrids H_0 and H_1 is in the signing queries. In the hybrid H_0 the adversary $\mathcal{A}$ can ask signing queries $\textsc{Sign}$ and in the hybrid H_1 the adversary $\mathcal{A}$ can ask signing queries $\textsc{Sign}'$. The answer to a $\textsc{Sign}'$ query is computed as follows.

For a sign query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ to P_j, check that the token tok_e has been previously been the answer to a query $(\textsc{Update}, F_e)$ and the public key pk_i has been previously output as an answer to a query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$. If this is the case, proceed as in the standard signing query $\textsc{Sign}$, otherwise, output $\perp$.

To show that the hybrids H_0 and H_1 are indistinguishable, it suffices to show that the probability that the adversary issues a signing query using a token tok_e or a receiver key pk_i that has not been previously generated by the challenger 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 either a token tok_e that verifies with respect to the signature scheme $\mathsf{DS}_{\mathsf{tok}}$ or a receiver key that verifies with respect to the signature scheme $\mathsf{DS}_{\mathsf{pub}}$, i.e., it needs to generate a key $\mathsf{pk}' := (\mathsf{vk}', P', \sigma'_{\mathsf{pub}})$ such that $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}', P'), \sigma'_{\mathsf{pub}}) = 1$ (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 either a forgery as captured by the event $\mathsf{PolicyForge}_{\mathcal{A}}$ or a key forgery as captured by the event $\mathsf{KeyForge}_{\mathcal{A}}$ in the proof of Theorem 5.1, and which can be defined and analyzed analogously here (with just minor syntactical changes).

Therefore, the event $\mathsf{SignForge}_{\mathcal{A}}$ is bounded by $\mathsf{KeyForge}$ and $\mathsf{PolicyForge}$, i.e., $\Pr[\mathsf{SignForge}_{\mathcal{A}}] \leq \Pr[\mathsf{KeyForge}_{\mathcal{A}}] + \Pr[\mathsf{PolicyForge}_{\mathcal{A}}]$, and the analysis of the events $\mathsf{KeyForge}_{\mathcal{A}}$ and $\mathsf{PolicyForge}_{\mathcal{A}}$ follows using the same reasoning as in Theorem 5.1 to conclude that $\Pr[\mathsf{KeyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}_0}(\lambda)$ and $\Pr[\mathsf{KeyForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{tok}},\mathcal{B}_1}(\lambda)$. This results in the fact that $\Pr[\mathsf{SignForge}_{\mathcal{A}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{pub}},\mathcal{B}_0}(\lambda) + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{tok}},\mathcal{B}_1}(\lambda)$, which proves the lemma. ☐

Lemma 5.5 (Indistinguishability of H_1 to H_2)**.** *Let* Π *be an adaptively UC-secure two-party computation protocol (Section 2.5), then the hybrids* H_1 *and* H_2 *are computationally indistinguishable.*

Proof. This statement follows by the UC composition theorem, as our security experiment can be seen as an environment for Π resp. $\mathcal{F}_C$ for computing circuit C. In particular, due to the security of Π against adaptive corruption, this allows us to handle adaptive corruption of the different parties during the protocol execution. ☐

Lemma 5.6 (Indistinguishability of H_2 to H_3)**.** *Let* NIZK *be non-interactive zero-knowledge proof, then the hybrids* H_2 *and* H_3 *are computationally indistinguishable.*

Proof. To show that the hybrids H_2 and H_3 are computationally indistinguishable, we build a reduction to the zero-knowledge property of the NIZK to argue the independence of the signatures from the attributes of the parties. Before we analyze this case, we stress that the policy update queries, key generation queries and corruption queries, are answered as in the previous hybrid. The only difference here is that the $\mathsf{CRS}_{\mathsf{NIZK}}$ is obtained from the underlying NIZK challenger and not generated by the reduction.

For a singing query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ asked to party P_j, we can now rely on $\mathcal{F}_C$ and the proof follows by a straightforward reduction. In more detail, in the ideal execution, the inputs of both, P_j and P_i, in the 2PC must be provided to the functionality (either by the reduction/environment or the adversary for corrupted parties). Let $(\mathsf{pk}_j, \mathsf{sk}_i := (\mathsf{vk}^R_{\mathsf{P},i}, \cdot, x_{i,0}, \sigma^i_{\mathsf{priv}}), \mathsf{tok}_e, \mathsf{mpk})$ denote the input of party P_i and $(\mathsf{pk}_i, \mathsf{sk}_j := (\mathsf{vk}^S_{\mathsf{P},j}, \cdot, x_{j,0}, \sigma^j_{\mathsf{priv}}), \mathsf{tok}_e, \mathsf{mpk})$ the input of party P_j. We can now evaluate the circuit as per specification $C_{\mathsf{CRS}_{\mathsf{NIZK}},\mathsf{vk}_{\mathsf{tok}},\mathsf{vk}_{\mathsf{priv}}}$ using the inputs of the parties P_j and P_i and, if all checks of the circuit succeed, we generate the proof π by forwarding $((F_e, \sigma_{\mathsf{tok}}), \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P},j}, \mathsf{vk}^R_{\mathsf{P},i}), (x_{j,0}, x_{i,0}, \sigma^j_{\mathsf{priv}}, \sigma^i_{\mathsf{priv}}))$ to the prove oracle of the underlying challenger for the NIZK protocol. If any of the checks does not succeed, we output $\perp$, otherwise, the proof π is used as an output. Afterwards, π is used to generate the final signature $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}^j_{\mathsf{P}}, (m, \mathsf{pk}_R, \pi))$ using the signing key $\mathsf{sk}^j_{\mathsf{P}}$ of party P_j and output $(m, \mathsf{pk}_i, \sigma := (\pi, \sigma'))$.

We observe that it follows from the zero-knowledge property that this emulation is perfect and, if the underlying challenger generates the proofs using the witness, then the hybrid H_2 is simulated and, if the underlying challenger simulates the proofs, then the hybrid H_3 is simulated. This concludes the proof of the lemma. ☐

5.2 Interactive UPCS using Predicate Encryption

The interactive scheme that we present in this section relies on the same primitives as the one from the previous section with the addition that it also requires a single-input predicate encryption scheme as well as an equivocal commitment scheme. The public and secret key pairs in this scheme have the same structure as in the previous scheme. In more detail, the public key consists of a verification key vk, whereas the secret key contains the corresponding signing key sk as well as the attributes x of the parties. To connect both of the keys and attest their authenticity a signature over vk as well as over (vk, x) is generated.

```
$\underline{\mathsf{Setup}(1^\lambda, F_{\mathsf{init}}):}$
$\mathsf{CRS}_{\mathsf{NIZK},1} \leftarrow \mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$
for $R^1_{\mathrm{ZK}}$ (Fig. 17).
$\mathsf{CRS}_{\mathsf{NIZK},2} \leftarrow \mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$
for $R^2_{\mathrm{ZK}}$ (Fig. 18).
$\mathsf{CRS}_{\mathsf{com}} \leftarrow \mathsf{com}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{mpk}_{\mathsf{PE}}, \mathsf{msk}_{\mathsf{PE}}) \leftarrow \mathsf{PE}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^\lambda)$
$\mathsf{sk}_{F_{\mathsf{init}}} \leftarrow \mathsf{PE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{PE}}, F_{\mathsf{init}})$
$\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}))$
$\mathsf{CRS}_{\mathsf{NIZK}} := \{\mathsf{CRS}_{\mathsf{NIZK},i}\}_{i \in [2]}$
$\mathsf{mpk} = (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{CRS}_{\mathsf{com}}, \mathsf{mpk}_{\mathsf{PE}},$
            $\mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$
$\mathsf{msk} := (\mathsf{msk}_{\mathsf{PE}}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{tok}})$
$\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$
Return $(\mathsf{mpk}, \mathsf{msk}), \mathsf{tok}_{\mathsf{init}}$

$\underline{\mathsf{PolUpd}(\mathsf{mpk}, \mathsf{msk}, F_{\mathsf{upd}})}$
Parse $\mathsf{mpk} := (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{CRS}_{\mathsf{com}}, \mathsf{mpk}_{\mathsf{PE}},$
                $\mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$
        $\mathsf{msk} := (\mathsf{msk}_{\mathsf{PE}}, \mathsf{sk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{tok}})$
Set $\widehat{F}_{\mathsf{upd}}$ as the single input function
that splits its input into two sets and
then evaluates $F_{\mathsf{upd}}$
$\mathsf{sk}_{F_{\mathsf{upd}}} \leftarrow \mathsf{PE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{PE}}, \widehat{F}_{\mathsf{upd}})$
$\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{tok}}, (F_{\mathsf{upd}}, \mathsf{sk}_{F_{\mathsf{upd}}}))$
$\mathsf{tok}_{\mathsf{upd}} := (F_{\mathsf{upd}}, \mathsf{sk}_{F_{\mathsf{upd}}}, \sigma'_{\mathsf{tok}})$
Return $\mathsf{tok}_{\mathsf{upd}}$

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

$\underline{\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma):}$
Parse $\mathsf{mpk} = (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{CRS}_{\mathsf{com}}, \mathsf{mpk}_{\mathsf{PE}},$
            $\mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$
      $\mathsf{tok} = (F, \mathsf{sk}_F, \sigma_{\mathsf{tok}}),$
      $\mathsf{pk}_S = (\mathsf{vk}_S, P_S, \sigma^S_{\mathsf{pub}}),$
      $\mathsf{pk}_R = (\mathsf{vk}_R, P_R, \sigma^R_{\mathsf{pub}}),$
      $\sigma = (\pi := (\mathsf{com}_{S,R}, \pi', \pi''), \sigma')$
(Return 0 if parsing fails or $\sigma = \perp$)
Return $\mathsf{Verify}(\mathsf{vk}_{\mathsf{tok}}, (F, \mathsf{sk}_F, \sigma_{\mathsf{tok}}))$
    $\wedge\ \mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_R, P_R), \sigma^R_{\mathsf{pub}})$
    $\wedge\ \mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}_S, P_S), \sigma^S_{\mathsf{pub}})$
    $\wedge\ \mathsf{Verify}(\mathsf{CRS}_{\mathsf{NIZK}}, (\mathsf{CRS}_{\mathsf{com}},$
            $\mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_S, \mathsf{vk}_R, \mathsf{com}_{S,R}), \pi')$
    $\wedge\ \mathsf{Verify}(\mathsf{CRS}_{\mathsf{NIZK}}, (\mathsf{CRS}_{\mathsf{com}},$
                $\mathsf{sk}_F, \mathsf{com}_{S,R}), \pi'')$
    $\wedge\ \mathsf{DS}_{\mathsf{P}}.\mathsf{Verify}(\mathsf{vk}_S, (m, \mathsf{pk}_R, \pi), \sigma')$
```

Figure 14: The setup, policy update, key generation and verification procedure of our interactive PE-based UPCS scheme.

For the generation of an update token for a policy F', the authority first turns this two-input policy, that takes as an input two attribute sets X_S and X_R, into a single-input function $\widehat{F}'$ that takes as an input a single set $X_{S,R}$ which is a disjoint union of the sender and receiver attributes, i.e., $X_{S,R} := \{(x, 0) | x \in X_S\} \cup \{(x, 1) | x \in X_R\}$. The function $\widehat{F}'$ splits the input set $X_{S,R}$ into two sets X_S and X_R, where attributes in X_S are of the form $(x, 0)$ and attributes in X_R are of the form $(x, 1)$, and then executes $F'(X_S, X_R)$. This transformation is needed since we only rely on a single-input predicate encryption scheme here. Afterwards, a functional key $\mathsf{sk}_{F'}$ for the policy $\widehat{F}'$ is generated and signed.

The first step of the signature generation works as in the previous scheme. In more detail, the sender and the receiver interact in a two-party computation protocol where the sender inputs its secret key $\mathsf{sk}_S := (\mathsf{vk}_S, x_S)$ and the public key of the receiver pk_R and the receiver inputs its secret key $\mathsf{sk}_R := (\mathsf{vk}_R, x_R)$ as well as the public key of the sender pk_S. Furthermore, both of the parties input the current token $(\mathsf{sk}_{F'}, \sigma_{\mathsf{tok}})$ as well as the master public key of the predicate encryption scheme $\mathsf{mpk}_{\mathsf{PE}}$. The circuit that the two-party computation protocol computes first verifies that the secret keys input by the parties correspond to the public keys, i.e., that sk_S corresponds to pk_S and sk_R corresponds to pk_R, by checking the contained information as well as verifying the signatures. Furthermore, it also verifies the signature of the token σ_{tok}. Afterwards, it generates a predicate encryption ciphertext $\mathsf{ct}_{S,R}$ encrypting a concatenation (x_S, x_R) of the attributes of the sender x_S and the receiver x_R as well as a commitment $\mathsf{com}_{S,R}$ to the ciphertext $\mathsf{ct}_{S,R}$. In the last step of the two-party computation protocol, a proof π' is generated that the commitment $\mathsf{com}_{S,R}$ commits to the ciphertext $\mathsf{ct}_{S,R}$ and that the ciphertext $\mathsf{ct}_{S,R}$ is honestly generated using the attributes x_S of the sender S and the attributes x_R of the receiver R. The statement used in this proof involves the commitment $\mathsf{com}_{S,R}$ as well as the verification keys of the sender vk_S and the receiver vk_R. Finally, the commitment $\mathsf{com}_{S,R}$ as well as its randomness r_{com}, the ciphertext $\mathsf{ct}_{S,R}$ and the proof π are output to the sender. The sender then generates the signature for a message m by first decrypting the obtained ciphertext $\mathsf{ct}_{S,R}$ using the functional key $\mathsf{sk}_{F'}$ of the current token and then proving, using a second zero-knowledge proof, that the commitment $\mathsf{com}_{S,R}$ commits to a ciphertext $\mathsf{ct}_{S,R}$ that decrypts to 1. The commitment $\mathsf{com}_{S,R}$ as well as the token $\mathsf{sk}_{F'}$ are used as the statement and the randomness for the commitment r_{com} and the ciphertext $\mathsf{ct}_{S,R}$ are used as the witness. The final step of the signature generation is by signing $(m, \mathsf{pk}_R, \pi := (\pi', \pi''))$, where π'' is the second proof generated as described above, using the signing key sk of the sender. For every further signature generation for the same pair of parties the ciphertext $\mathsf{ct}_{S,R}$ and the commitment $\mathsf{com}_{S,R}$, and therefore the resulting proofs π' and π'', in case that the proof is generated under the same policy, can be reused. If a signature for the same pair of parties needs to be generated for a different token, a new proof π'' needs to be generated and the remaining information can be reused.

The commitment here is needed to ensure that the ciphertext is not revealed when a signature is generated. A revelation of the ciphertext would allow every other party in the system to evaluate every policy over the attributes of the parties that are involved in this single signature generation. Another alternative to prevent this leakage, if someone does not want to rely on a commitment scheme, is to generate the second proof for the relation that proves that the owned ciphertext decrypts to 1 and that the proof output by the two-party computation protocol verifies. This then requires the generation of a proof of a proof which might be less efficient than a second proof that only proves some properties of the commitment. Therefore, we present the construction using the commitment scheme.

To verify a signature, the signatures associated with the public keys of the sender and receiver are verified, as well as the signature on the token for which it has been generated. If these checks succeed, then the information in the public keys and the token can be used to verify the proofs π' and π'' of the signature. Lastly, the signature σ generated over the proof and the message $(m, \mathsf{pk}_R, \pi := (\pi', \pi''))$ is verified using the sender's verification key vk_S. If all of these verifications succeed, the signature is deemed valid. We describe our formal scheme in Figures 14 and 15.

```
$\Pi_{\text{sign}}(\mathsf{mpk}, \mathsf{tok}, \mathsf{sk}_S, \mathsf{pk}_S, R, m)$ for signer $S$:
Parse $\mathsf{mpk} = (\mathsf{CRS}_{\mathsf{NIZK}}, \mathsf{CRS}_{\mathsf{com}}, \mathsf{mpk}_{\mathsf{PE}}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}}), \mathsf{tok} = (F, \mathsf{sk}_F, \sigma_{\mathsf{tok}})$,
        $\mathsf{sk}_S = (\mathsf{vk}^S_{\mathsf{P}}, \mathsf{sk}^S_{\mathsf{P}}, x_S, \sigma^S_{\mathsf{priv}}), \mathsf{pk}_S = (\mathsf{vk}^S_{\mathsf{P}}, P_S, \sigma^S_{\mathsf{pub}})$
Obtain $\mathsf{pk}_R = (\mathsf{vk}^R_{\mathsf{P}}, P_R, \sigma^R_{\mathsf{pub}})$ from $R$ (cf. Sec. 3)
If the algorithms were already executed between $S$ and $R$, proceed to the decryption step.
Else:
$S$: Return $\perp$, if $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}^R_{\mathsf{P}}, P_R), \sigma^R_{\mathsf{pub}}) = 0$.
$R$: Return $\perp$, if $\mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{pub}}, (\mathsf{vk}^S_{\mathsf{P}}, P_S), \sigma^S_{\mathsf{pub}}) = 0$.
$S$ and $R$ execute $\Pi$ that computes the circuit described in Figure 16.
After the execution of $\Pi$, $S$ obtains $(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R}, r_{\mathsf{com}}, \pi')$.
If $\mathsf{PE}.\mathsf{Dec}(\mathsf{sk}_F, \mathsf{ct}_{S,R}) = 0$, return $\perp$
$S$ computes $\pi'' \leftarrow \mathsf{NIZK}_2.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{NIZK},2}, (\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_F, \mathsf{com}_{S,R}), (\mathsf{ct}_{S,R}, r_{\mathsf{com}}))$ for the
relation $R^2_{\mathsf{ZK}}$ described in Figure 18 and erases the randomness used for the generation of $\pi''$.
Compute $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}^S_{\mathsf{P}}, (m, \mathsf{pk}_R, \mathsf{com}_{S,R}, \pi := (\pi', \pi'')))$, set $\sigma := (\mathsf{com}_{S,R}, \pi, \sigma')$.
Return $(m, \mathsf{pk}_R, \sigma)$.
```

Figure 15: The signing procedure of our interactive PE-based UPCS scheme.

```
Circuit $C_{\mathsf{CRS}_{\mathsf{NIZK},1}, \mathsf{mpk}_{\mathsf{PE}}, \mathsf{vk}_{\mathsf{priv}}}$:
Inputs: $S$ uses $(\mathsf{pk}_R := (\mathsf{vk}^R_{\mathsf{P},S}, P_R, \sigma^R_{\mathsf{pub},S}), \mathsf{sk}_S := (\mathsf{vk}^S_{\mathsf{P},S}, \cdot, x_S, \sigma^S_{\mathsf{priv}})$,
            $\mathsf{mpk}_S := (\mathsf{CRS}^S_{\mathsf{NIZK}}, \mathsf{CRS}^S_{\mathsf{com}}, \mathsf{mpk}^S_{\mathsf{PE}}, \cdot, \mathsf{vk}^S_{\mathsf{priv}}, \cdot))$ as its input and
        $R$ uses $(\mathsf{pk}_S := (\mathsf{vk}^S_{\mathsf{P},R}, P_S, \sigma^S_{\mathsf{pub},R}), \mathsf{sk}_R := (\mathsf{vk}^R_{\mathsf{P},R}, \cdot, x_R, \sigma^R_{\mathsf{priv}})$,
            $\mathsf{mpk}_R := (\mathsf{CRS}^R_{\mathsf{NIZK}}, \mathsf{CRS}^R_{\mathsf{com}}, \mathsf{mpk}^R_{\mathsf{PE}}, \cdot, \mathsf{vk}^R_{\mathsf{priv}}, \cdot))$ as its input.
        Check that:
        • $\mathsf{CRS}_{\mathsf{NIZK}} := \mathsf{CRS}^S_{\mathsf{NIZK}} = \mathsf{CRS}^R_{\mathsf{NIZK}}, \mathsf{CRS}_{\mathsf{com}} := \mathsf{CRS}^S_{\mathsf{com}} = \mathsf{CRS}^R_{\mathsf{com}}$,
          $\mathsf{mpk}_{\mathsf{PE}} := \mathsf{mpk}^S_{\mathsf{PE}} = \mathsf{mpk}^R_{\mathsf{PE}}, \mathsf{vk}_{\mathsf{priv}} := \mathsf{vk}^S_{\mathsf{priv}} = \mathsf{vk}^R_{\mathsf{priv}}$.
        • $\mathsf{vk}^R_{\mathsf{P}} := \mathsf{vk}^R_{\mathsf{P},S} = \mathsf{vk}^R_{\mathsf{P},R}$ and $\mathsf{vk}^S_{\mathsf{P}} := \mathsf{vk}^S_{\mathsf{P},R} = \mathsf{vk}^S_{\mathsf{P},S}$.
        • $\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}^S_{\mathsf{P}}, x_S), \sigma^S_{\mathsf{priv}}) = 1$ and $\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}^R_{\mathsf{P}}, x_R), \sigma^R_{\mathsf{priv}}) = 1$.
        If any of these checks fails, output $\perp$.
        If all of these checks are successful, compute:
        • $\mathsf{ct}_{S,R} := \mathsf{PE}.\mathsf{Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_S \dot{\cup} x_R); r_{\mathsf{Enc}})$ with $r_{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$ and "$\dot{\cup}$" as in Section 2.
        • $\mathsf{com}_{S,R} := \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, \mathsf{ct}_{S,R}; r_{\mathsf{com}})$, with $r_{\mathsf{com}} \leftarrow \{0,1\}^\lambda$.
        • $\pi' \leftarrow \mathsf{NIZK}_1.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{NIZK},1}, (\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P}}, \mathsf{vk}^R_{\mathsf{P}}, \mathsf{com}_{S,R})$,
                        $(x_S, x_R, \sigma^S_{\mathsf{priv}}, \sigma^R_{\mathsf{priv}}, \mathsf{ct}_{S,R}, r_{\mathsf{Enc}}, r_{\mathsf{com}}))$
          for the relation $R^1_{\mathsf{ZK}}$ described in Figure 17.
Output $(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R}, r_{\mathsf{com}}, \pi')$ to $S$.
```

Figure 16: The circuit computed by the 2PC in the signing procedure of the interactive UPCS scheme in Figure 15.

Relation R^1_{ZK}:
Instance: $x = (\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P}}, \mathsf{vk}^R_{\mathsf{P}}, \mathsf{com}_{S,R})$
Witness: $w = (x_S, x_R, \sigma^S_{\mathsf{priv}}, \sigma^R_{\mathsf{priv}}, \mathsf{ct}_{S,R}, r_{\mathsf{Enc}}, r_{\mathsf{com}})$
$R_{\mathsf{ZK}}(x, w) = 1$ if and only if:
$\mathsf{DS}_{\mathsf{priv}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{priv}}, (\mathsf{vk}^P_{\mathsf{P}}, x_P), \sigma^P_{\mathsf{priv}}) = 1$, for $P \in \{S, R\}$,
$\mathsf{ct}_{S,R} \coloneqq \mathsf{PE}.\mathsf{Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_S \dot{\cup} x_R); r_{\mathsf{Enc}})$ and $\mathsf{com}_{S,R} \coloneqq \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, \mathsf{ct}_{S,R}; r_{\mathsf{com}})$.

Figure 17: Relation used for the NIZK inside the 2PC of the interactive UPCS scheme in Figure 16.

Relation R^2_{ZK}:
Instance: $x = (\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_F, \mathsf{com}_{S,R})$
Witness: $w = (\mathsf{ct}_{S,R}, r_{\mathsf{com}})$
$R_{\mathsf{ZK}}(x, w) = 1$ if and only if:
$\mathsf{com} \coloneqq \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, \mathsf{ct}_{S,R}, r_{\mathsf{com}})$ and $\mathsf{PE}.\mathsf{Dec}(\mathsf{sk}_F, \mathsf{ct}_{S,R}) = 1$

Figure 18: Relation used for the final NIZK in the signature generation of the interactive UPCS scheme in Figure 15.

5.2.1 PE Instantiations

For the interactive UPCS scheme described above, we require attribute-hiding PE with adaptive security. This primitive has been realized for *inner-product* predicates in the standard model based on bilinear pairings in [OT12]. In more detail, such an instantiation allows to evaluate expressions such as $\langle(\boldsymbol{x}, \boldsymbol{y}), (\boldsymbol{u}, \boldsymbol{v})\rangle = (\langle \boldsymbol{x}, \boldsymbol{u}\rangle + \langle \boldsymbol{y}, \boldsymbol{v}\rangle) \bmod \mathbb{Z}_q$, where $\boldsymbol{x} \in \mathbb{Z}_q^n$ and $\boldsymbol{y} \in \mathbb{Z}_q^n$ are the attributes associated with the sender and receiver respectively, and $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{Z}_q^{2n}$ defines the policy (for any $n \in \mathbb{N}$ and some prime q). The values $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}$ and $\boldsymbol{v}$ here might be obtained by passing the attributes of the sender and the receiver, as well as the policy, through a *preprocessing* phase. Such a preprocessing procedure can be defined similarly as in [KSW08, Section 5] and [BMW21, Section 4.5] to obtain more concrete policies.

Further, we can also instantiate the underlying PE scheme with an adaptively secure, attribute-hiding PE scheme for *quadratic* predicates in the generic group model, again, based on pairings, from [BCFG17, Section 6] or [RPB+19, Section 3]. Concretely, having $\boldsymbol{x} \in \mathbb{Z}_q^{n_1}$ and $\boldsymbol{y} \in \mathbb{Z}_q^{n_2}$ (for some $n_1, n_2 \in \mathbb{N}$) as the sender's and receiver's attributes respectively, the policies P are described by matrices $\mathbf{F} \in \mathbb{Z}_q^{n_1 \times n_2}$ as $P((\boldsymbol{x}, \boldsymbol{y}), \mathbf{F}) = \left(\boldsymbol{x}^\top \mathbf{F} \boldsymbol{y}\right) \bmod \mathbb{Z}_q$. Such policies also capture richer functions like constant-depth boolean formulas or comparison predicates [BCFG17, Section 6.1].

These pairing-based PE schemes show that our UPCS scheme with one-time interaction can, in principle, be implemented and benchmarked supporting different classes of predicates and given efficient instantiations of digital signatures and NIZK proofs that are compatible with the underlying PE schemes.

5.2.2 Correctness and Security

The correctness of the scheme follows directly from the correctness of the underlying schemes: DS, NIZK, 2PC and that of PE. For security, unforgeability follows from the unforgeability of the signature schemes, the (knowledge) soundness of the NIZK proof and the binding property of the commitment com. Further, the attribute-hiding of this UPCS schemes follows from the simulatability of the 2PC, the zero-knowledge property of NIZK and the attribute-hiding of PE. We present the formal proofs below. We note in passing that when switching to a model where we cannot erase the randomness during proof generation, we obtain an analogous result when switching to a NIZK that supports adaptive corruptions [GOS06].

Unforgeability

Now, we prove the unforgeability of our construction.

Theorem 5.7. *Let* $\mathsf{DS_{pub}} = (\mathsf{DS_{pub}.Setup}, \mathsf{DS_{pub}.Sign}, \mathsf{DS_{pub}.Verify})$*,* $\mathsf{DS_{tok}} = (\mathsf{DS_{tok}.Setup}, \mathsf{DS_{tok}.Sign}, \mathsf{DS_{tok}.Verify})$*,* $\mathsf{DS_{priv}} = (\mathsf{DS_{priv}.Setup}, \mathsf{DS_{priv}.Sign}, \mathsf{DS_{priv}.Verify})$ *and* $\mathsf{DS_P} = (\mathsf{DS_P.Setup}, \mathsf{DS_P.Sign}, \mathsf{DS_P.Verify})$ *be EUF-CMA secure signature schemes, let* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *be an extractable proof system and* Π *be an adaptively UC-secure two-party computation protocol for the specified circuit in Figure 16 and* $\mathsf{com} = (\mathsf{Setup}, \mathsf{Com})$ *an equivocal commitment scheme, then the construction* $\mathsf{UPCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figures 14 and 15, is existentially unforgeable.*

Proof. Using the composition theorem, we can effectively work in a world where parties use $\mathcal{F}_C$ to evaluate a circuit C and rely on the correctness of the computation to ensure the computed values are as specified. The proof of this theorem proceeds very similar to the proof of the non-interactive scheme. We define the same events as in the proof of Theorem 4.3. We recap them here:

- Event $\mathsf{KeyForge}_\mathcal{A}$: The adversary $\mathcal{A}$ terminates with output $(\mathsf{tok}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ where the public key pk_S (resp. pk_R) does not belong to any initialized party P_S (resp. P_R) (by means of an invocation $(\textsc{KeyGen}, P_S, x_S)$, (resp. $(\textsc{KeyGen}, P_R, x_R)$)).
- Event $\mathsf{KeyColl}_\mathcal{A}$: The adversary terminates and it holds that there are two parties P_i and P_j where $i \neq j$ such that for the corresponding public keys $\mathsf{pk}_i = (\mathsf{vk}_i, \cdot, \cdot, \cdot)$ and $\mathsf{pk}_j = (\mathsf{vk}_j, \cdot, \cdot, \cdot)$ it holds that $\mathsf{vk}_i = \mathsf{vk}_j$.

Denote the winning condition of the experiment by an event $\mathsf{WIN}_\mathcal{A}$ and split it into three parts:

- Event $\mathsf{WIN1}_\mathcal{A}$: The adversary generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where the public key pk is associated with some initialized party P_S that is not corrupted and was never invoked on input $(\textsc{Sign}, P_S, \mathsf{tok}, \mathsf{pk}_R, m)$.
- Event $\mathsf{WIN2}_\mathcal{A}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where the update token tok^* is not associated with a policy F^* that has never been an input to the authority $(\textsc{Update}, F^*)$. We also denote this event as $\mathsf{PolicyForge}_\mathcal{A}$.
- Event $\mathsf{WIN3}_\mathcal{A}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where there is a policy F' such that a query $(\textsc{Update}, F')$ has been asked, and there are attributes x_S and x_R associated to some initialized parties P_S and P_R with public keys pk_S and pk_R, respectively (as the result of inputs $(\textsc{KeyGen}, P_S, x_S)$ and $(\textsc{KeyGen}, P_R, x_R)$ to $\mathfrak{A}$), such that $F'(x_S, x_R) = 0$.

The bounds for events $\mathsf{KeyForge}_\mathcal{A}$ and $\mathsf{KeyColl}_\mathcal{A}$ follows as in the non-interactive scheme, i.e. by Lemmata 4.4 and 4.5:

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

for adversaries $\mathcal{B}_1$ and $\mathcal{B}'_2$ which are constructed based on $\mathcal{A}$ and have roughly the same efficiency as $\mathcal{A}$ and where q are the number of key generation queries KeyGen.

Also the bounds of the events $\mathsf{WIN1}_\mathcal{A}, \mathsf{WIN2}_\mathcal{A}$ and $\mathsf{WIN3}_\mathcal{A}$ have the same bounds as in the non-interactive case.

$$\begin{aligned}
&\Pr[\mathsf{WIN1}_\mathcal{A}] \quad \leq q \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_\mathsf{P},\mathcal{B}''_2}(\lambda),\\
&\Pr[\mathsf{WIN2}_\mathcal{A} / \ \mathsf{PolicyForge}_\mathcal{A}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_\mathsf{tok},\mathcal{B}'_1}(\lambda) \text{ and}\\
&\Pr[\mathsf{WIN3}_\mathcal{A} \cap \overline{\mathsf{KeyColl}_\mathcal{A} \cup \mathsf{KeyForge}_\mathcal{A} \cup \mathsf{PolicyForge}_\mathcal{A}}]\\
&\qquad\qquad \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_\mathsf{priv},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK}_1,\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK}_1,\mathcal{B}'}\\
&\qquad\qquad\quad + \mathsf{Adv}^{\text{BIND}}_{\mathsf{Com},\mathcal{B}_5}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK}_2,\mathcal{B}_6}(\lambda) + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK}_2,\mathcal{B}''},
\end{aligned}$$

where the adversaries $\mathcal{B}'_1, \mathcal{B}''_2$, $\mathcal{B}_3$, $\mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_6$, and distinguishers $\mathcal{B}'$ and $\mathcal{B}''$ are constructed based on $\mathcal{A}$ and have roughly the same efficiency as $\mathcal{A}$.

The proof of the bounds for the events $\mathsf{WIN1}_\mathcal{A}$ and $\mathsf{WIN2}_\mathcal{A}$ proceeds in almost exactly the same way as described in the proofs of Lemmata 4.6 and 4.7. The proof of the bound for $\mathsf{WIN3}_\mathcal{A}$ differs slightly from the proof in the non-interactive case, therefore, we recap the adjusted proof in Lemma 5.8.

By definition of the events, we have

$$\begin{aligned}
\Pr[\mathsf{WIN}_\mathcal{A}] \leq &\Pr[\mathsf{KeyColl}_\mathcal{A} \cup \mathsf{KeyForge}_\mathcal{A} \cup \mathsf{PolicyForge}_\mathcal{A}]\\
&+ \Pr[\mathsf{WIN}_\mathcal{A} \cap \overline{\mathsf{KeyColl}_\mathcal{A} \cup \mathsf{KeyForge}_\mathcal{A} \cup \mathsf{PolicyForge}_\mathcal{A}}]\\
\leq &\Pr[\mathsf{KeyColl}_\mathcal{A}] + \Pr[\mathsf{KeyForge}_\mathcal{A}] + \Pr[\mathsf{PolicyForge}_\mathcal{A}]\\
&+ \Pr[\mathsf{WIN1}_\mathcal{A} \ \cap \ \overline{\mathsf{KeyColl}_\mathcal{A} \cup \mathsf{KeyForge}_\mathcal{A} \cup \mathsf{PolicyForge}_\mathcal{A}}]\\
&+ \underbrace{\Pr[\mathsf{WIN2}_\mathcal{A} \ \cap \ \overline{\mathsf{KeyColl}_\mathcal{A} \cup \mathsf{KeyForge}_\mathcal{A} \cup \mathsf{PolicyForge}_\mathcal{A}}]}_{=0, \text{ since } \mathsf{WIN2}_\mathcal{A} = \mathsf{PolicyForge}_\mathcal{A}}\\
&+ \Pr[\mathsf{WIN3}_\mathcal{A} \ \cap \ \overline{\mathsf{KeyColl}_\mathcal{A} \cup \mathsf{KeyForge}_\mathcal{A} \cup \mathsf{PolicyForge}_\mathcal{A}}].
\end{aligned}$$

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$^{\mathsf{DS}_\mathsf{P}}$. Therefore, the theorem follows. □

Lemma 5.8. *Consider the unforgeability experiment and let* $\mathsf{WIN3}_\mathcal{A}$ *be defined as above. We can construct adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *and a distinguisher* $\mathcal{B}'$ *such that*

$$\Pr[\mathsf{WIN3}_\mathcal{A} \cap \overline{\mathsf{KeyColl}_\mathcal{A} \cup \mathsf{KeyForge}_\mathcal{A} \cup \mathsf{PolicyForge}_\mathcal{A}}] \leq \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_\mathsf{priv},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda).$$

Proof. To prove this theorem, we split the event $\mathsf{WIN3}_\mathcal{A}$ into two different events $\mathsf{WIN3.1}_\mathcal{A}$ and $\mathsf{WIN3.2}_\mathcal{A}$ which are defined as follows:

$\mathsf{WIN3.1}_\mathcal{A}$**:** The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_\mathsf{tok}), \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^* := (\mathsf{com}_{S,R}, \pi := (\pi', \pi''), \sigma'))$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where there is a policy F' such that a query (Update, F') has been asked, and there are attributes x_S and x_R associated to some initialized parties P_S and P_R with public keys pk_S and pk_R, respectively (as the result of inputs $(\text{KeyGen}, P_S, x_S)$ and $(\text{KeyGen}, P_R, x_R)$ to $\mathfrak{A}$), such that $F'(x_S, x_R) = 0$ where it holds that $R^2_{\mathrm{ZK}}((\mathsf{CRS}_\mathsf{com}, \mathsf{sk}_{F^*}, \mathsf{com}_{S,R}), (\mathsf{ct}_{S,R}, r_\mathsf{com})) = 1$ with $\mathsf{ct}_{S,R}, \mathsf{com}_{S,R}$ and r_com being output by the two-party computation protocol between S and R.

$\mathsf{WIN3.2}_{\mathcal{A}}$**:** The adversary $\mathcal{A}$ generates the output $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_{\mathsf{tok}}), \mathsf{pk} := (\mathsf{vk}^*_{\mathsf{P},S}, P^*_S, \sigma^*_{\mathsf{pub},S}),$ $\mathsf{pk}^* := (\mathsf{vk}^*_{\mathsf{P},R}, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\mathsf{com}_{S,R}, \pi := (\pi', \pi''), \sigma'))$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1$ and where there is a policy F' such that a query $(\textsc{Update}, F')$ has been asked, and there are attributes x_S and x_R associated to some initialized parties P_S and P_R with public keys pk_S and pk_R, respectively (as the result of inputs $(\textsc{KeyGen}, P_S, x_S)$ and $(\textsc{KeyGen}, P_R, x_R)$ to $\mathfrak{A}$), such that $F'(x_S, x_R) = 0$ where it holds that $R^1_{\mathsf{ZK}}((\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P}}, \mathsf{vk}^R_{\mathsf{P}}, \mathsf{com}_{S,R}), (x_S, x_R, \sigma^S_{\mathsf{priv}}, \sigma^R_{\mathsf{priv}}, \mathsf{ct}_{S,R}, r_{\mathsf{Enc}}, \mathsf{com}_{S,R}, r_{\mathsf{com}})) = 1$ with $(x_S, \sigma^S_{\mathsf{priv}})$ and $(x_R, \sigma^R_{\mathsf{priv}})$ being parts of the secret keys corresponding to pk and pk^* respectively and where $\mathsf{ct}_{S,R}, \mathsf{com}_{S,R}$ and $r_{\mathsf{Enc}}, r_{\mathsf{com}}$ are generated by the two-party computation protocol and $\mathsf{vk}_{\mathsf{priv}}$ is part of the master public key.

From the definition of the events $\mathsf{WIN3.1}_{\mathcal{A}}$ and $\mathsf{WIN3.2}_{\mathcal{A}}$, it follows that $\mathsf{WIN3}_{\mathcal{A}} = \mathsf{WIN3.1}_{\mathcal{A}} \cup \mathsf{WIN3.2}_{\mathcal{A}}$ which, in turn, implies that $\Pr[\mathsf{WIN3}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}] = \Pr[\mathsf{WIN3.1}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}] + \Pr[\mathsf{WIN3.2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}]$. For the first event $\mathsf{WIN3.1}_{\mathcal{A}}$, it holds that $\Pr[\mathsf{WIN3.1}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}] \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{DS}_{\mathsf{priv}},\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK}_1,\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK}_1,\mathcal{B}'}$. The proof for this bound follows using the arguments of Lemmata 4.8 and 5.8. Therefore, we refer to these Lemmas for the proof and focus in the remainder of this proof on bounding the event $\mathsf{WIN3.2}_{\mathcal{A}}$.

Lemma 5.9. *Consider the unforgeability experiment and let* $\mathsf{WIN3.2}_{\mathcal{A}}$ *be defined as above. We can construct adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *and a distinguisher* $\mathcal{B}'$ *such that*

$$\Pr[\mathsf{WIN3.2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}] \leq \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + \mathsf{Adv}^{\text{BIND}}_{\mathsf{Com},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda).$$

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 binding of the underlying commitment scheme in order to possess a valid witness.

We first make a first transition to a hybrid world $\text{EUF-CMA}^{\mathsf{UPCS}}_{\text{Hyb}}$, which is identical to $\text{EUF-CMA}^{\mathsf{UPCS}}$ except that we replace $\mathsf{NIZK}_1.\mathsf{Setup}(1^\lambda)$ $\mathsf{NIZK}_2.\mathsf{Setup}(1^\lambda)$ by the CRS simulation algorithm E_1 associated to the NIZK schemes. All above defined events are still defined in this hybrid experiment. Clearly, we can construct a distinguisher $\mathcal{B}'$ such that

$$\begin{aligned}\Pr[\mathsf{WIN3.2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}] \\ \leq \Pr_{\text{Hyb}}\left[\mathsf{WIN3.2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right] \\ + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK}_1,\mathcal{B}'} + \mathsf{Adv}^{\text{CRS}}_{\mathsf{NIZK}_2,\mathcal{B}''},\end{aligned}$$

where $\Pr_{\text{Hyb}}[.]$ makes explicit that this probability is taken w.r.t. experiment $\text{EUF-CMA}^{\mathsf{UPCS}}_{\text{Hyb}}$. This reduction is standard: in order to distinguish the two distributions, on input a sample CRS, the distinguisher $\mathcal{B}'$, and $\mathcal{B}'$ respectively, emulates the experiment towards $\mathcal{A}$. When $\mathcal{A}$ terminates, $\mathcal{B}'$ (or $\mathcal{B}''$) outputs 1 if event $\mathsf{WIN3.2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}$ occurs (which is computable by $\mathcal{B}'/\mathcal{B}''$ that manages all key-sets).

We build an adversary $\mathcal{B}_1$ that simulates $\text{EUF-CMA}^{\mathsf{UPCS}}_{\text{Hyb}}$ towards $\mathcal{A}$ when interacting with the underlying $\text{BIND}^{\mathsf{Com}}$ experiment. We show that if $\mathcal{A}$ outputs $(\mathsf{tok}^*, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ as event $\mathsf{WIN3.2}_{\mathcal{A}}$ defines it can be used as a binding attack in the $\text{BIND}^{\mathsf{Com}}$ 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 $\mathsf{CRS}_{\mathsf{com}}$ from the underlying challenger and the policy F_{init} from the adversary $\mathcal{A}$. In the next step, $\mathcal{B}_1$ generates $\mathsf{CRS}_{\mathsf{NIZK},1} \leftarrow \mathsf{NIZK}.E_1(1^\lambda)$, $\mathsf{CRS}_{\mathsf{NIZK},2} \leftarrow \mathsf{NIZK}_2.E_1(1^\lambda)$, $(\mathsf{vk}_{\mathsf{pub}}, \mathsf{sk}_{\mathsf{pub}}) \leftarrow \mathsf{DS}_{\mathsf{pub}}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{priv}}, \mathsf{sk}_{\mathsf{priv}}) \leftarrow \mathsf{DS}_{\mathsf{priv}}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{tok}}, \mathsf{sk}_{\mathsf{tok}}) \leftarrow$

$\mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}(1^\lambda)$, $(\mathsf{mpk}_{\mathsf{PE}}, \mathsf{msk}_{\mathsf{PE}}) \leftarrow \mathsf{PE}.\mathsf{Setup}(1^\lambda)$ and sets $\mathsf{mpk} := (\mathsf{CRS}_{\mathsf{NIZK}} := \{\mathsf{CRS}_{\mathsf{NIZK},1}, \mathsf{CRS}_{\mathsf{NIZK},2}\}$, $\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{pub}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{tok}})$. Furthermore, $\mathcal{B}$ generates $\mathsf{sk}_{F_{\mathsf{init}}} \leftarrow \mathsf{PE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{PE}}, \mathsf{sk}_{F_{\mathsf{init}}})$ and signs it $\sigma_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}(\mathsf{sk}_{\mathsf{tok}}, (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}))$. Afterwards, it sets $\mathsf{tok}_{\mathsf{init}} := (F_{\mathsf{init}}, \mathsf{sk}_{F_{\mathsf{init}}}, \sigma_{\mathsf{tok}})$ and sends mpk and $\mathsf{tok}_{\mathsf{init}}$ to the adversary $\mathcal{A}$. The adversary $\mathcal{B}$ also initializes the counter $e = 2$.

If the adversary $\mathcal{A}$ asks a policy update query $(\textsc{Update}, F')$, the adversary $\mathcal{B}$ generates $\mathsf{sk}_{F'} \leftarrow \mathsf{PE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{PE}}, F')$, signs it $\sigma'_{\mathsf{tok}} \leftarrow \mathsf{DS}_{\mathsf{tok}}(\mathsf{sk}_{\mathsf{tok}}, (F', \mathsf{sk}_{F'}))$ and outputs $\mathsf{tok}' := (F', \mathsf{sk}_{F'}, \sigma'_{\mathsf{tok}})$ to $\mathcal{A}$. Afterwards, it increases e, i.e. $e := e + 1$.

For a key generation query $(\textsc{KeyGen}, P_i, x)$ to the key generation oracle $\mathsf{QKeyGen}$, $\mathcal{B}_1$ samples a signature key pair $(\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \leftarrow \mathsf{DS}.\mathsf{Setup}(1^\lambda)$ and generates the signatures $\sigma_{\mathsf{pub}} \leftarrow \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{pub}}, (\mathsf{vk}_{\mathsf{P}}, P_i))$ and submits as well as $\sigma_{\mathsf{priv}} \leftarrow \mathsf{DS}_{\mathsf{priv}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{priv}}, (\mathsf{vk}_{\mathsf{P}}, x))$. Then, $\mathcal{B}_1$ sets $\mathsf{pk} := (\mathsf{vk}_{\mathsf{P}}, P_i, \sigma_{\mathsf{pub}})$ and $\mathsf{sk} := (\mathsf{vk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}, x, \sigma_{\mathsf{priv}})$ 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 $(\textsc{Corrupt}, P_j)$, the adversary $\mathcal{B}_1$ searches for the key sk that is associated with P_j. If no such entry exists, the adversary $\mathcal{B}_1$ outputs $\perp$ to $\mathcal{A}$, otherwise it sends sk to $\mathcal{A}$.

If the adversary $\mathcal{A}$ asks a signing query $(\textsc{Sign}, \mathsf{tok}, P_j, \mathsf{pk}_R := (\mathsf{vk}_R, P_R, \sigma^R_{\mathsf{pub}}), m)$, then the adversary $\mathcal{B}_1$ checks that a key has been generated for P_j and that pk_R has been the reply to a previous key generation query $\textsc{KeyGen}$. If this is not the case, the adversary $\mathcal{B}_1$ aborts. Otherwise, it continues with the execution of Sign, using the key sk of party P_j and terminates once Sign terminates.

When $\mathcal{A}$ terminates with $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_{\mathsf{tok}}), \mathsf{pk}^*_S := (\mathsf{vk}^*_{\mathsf{P},S}, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_{\mathsf{P},R}, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^* := (\pi', \pi''), \sigma'))$ $\mathcal{B}_1$ 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 $\mathsf{WIN3.2}_{\mathcal{A}}$ hold and in case this is true, it first calls $(x^*_S, x^*_R, \sigma^*_{\mathsf{priv},S}, \sigma^*_{\mathsf{priv},R}, \mathsf{ct}_{S,R}, r_{\mathsf{Enc}}, r_{\mathsf{com}}) \leftarrow E_2(\mathsf{CRS}_{\mathsf{NIZK},1}, (\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{vk}^*_{\mathsf{P},R}, \mathsf{com}_{S,R}), \pi')$ and $(\mathsf{ct}'_{S,R}, r'_{\mathsf{com}}) \leftarrow E_2(\mathsf{CRS}_{\mathsf{NIZK},2}, (\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_{F^*}, \mathsf{com}_{S,R}), \pi'')$ and checks whether $(x := (\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^*_{\mathsf{P},S}, \mathsf{vk}^*_{\mathsf{P},R}, \mathsf{com}_{S,R}), w := (x^*_S, x^*_R, \sigma^*_{\mathsf{priv},S}, \sigma^*_{\mathsf{priv},R}, \mathsf{ct}_{S,R}, r_{\mathsf{Enc}}, r_{\mathsf{com}})) \in R^1_{\mathrm{ZK}}$ and $(x := (\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_{F^*}, \mathsf{com}_{S,R}), w := (\mathsf{ct}'_{S,R}, r'_{\mathsf{com}})) \in R^2_{\mathrm{ZK}}$ (which is efficiently checkable) and if this is the case, it submits $(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R}, r_{\mathsf{com}}, \mathsf{ct}'_{S,R}, r'_{\mathsf{com}})$, as a forgery to the underlying binding experiment $\mathrm{BIND}^{\mathsf{Com}}$. If $(x, w) \notin R^1_{\mathrm{ZK}}$ or $(x', w') \notin R^2_{\mathrm{ZK}}$ 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 $\mathrm{EUF\text{-}CMA}^{\mathsf{UPCS}}_{\mathrm{Hyb}}$ 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 $\mathcal{A}$ have been previously been an answer to a query $\textsc{KeyGen}$. Therefore, there are parties P_j and P_k such that pk^*_S is the key of P_j and pk^*_R is the key of P_k and where the keys contain vk_i and are associated with x_i for $i \in \{j, k\}$. Let us fix these two indices. Furthermore, we can assume that $\mathsf{NIZK}.\mathsf{Verify}(\mathsf{CRS}, (\mathsf{tok}^*, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_j, \mathsf{vk}_k), \pi^*) = 1$ as otherwise, $\mathsf{WIN3.2}_{\mathcal{A}}$ does not hold. Additionally, we know that $F^*(x_j, x_k) = 0$ and that the two-party computation has been executed correctly, assuming the extractability of NIZK_1 which we argue below, it follows that $\mathsf{ct}_{S,R} \neq \mathsf{ct}'_{S,R}$ and $r_{\mathsf{com}} \neq r'_{\mathsf{com}}$. Therefore, it holds that $(\mathsf{ct}_{S,R}, r_{\mathsf{com}})$ and $(\mathsf{ct}'_{S,R}, r'_{\mathsf{com}})$ is a valid opening for the commitment $\mathsf{com}_{S,R}$ which results in a valid binding attack. We obtain that the probability that $\mathcal{B}_1$ terminates with a valid forgery is therefore

$$\mathsf{Adv}^{\mathrm{BIND}}_{\mathsf{Com},\mathcal{B}_1} = \Pr_{\mathrm{Hyb}}[\mathsf{WIN3.2}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\mathrm{ext}}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}].$$

We obtain

$$\begin{aligned}\Pr_{\mathrm{Hyb}}\left[\mathsf{WIN3.2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right] \\ = \mathsf{Adv}^{\mathrm{BIND}}_{\mathsf{Com},\mathcal{B}_1} + \mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK}_1,\mathcal{B}_2} + \\ \underbrace{\Pr_{\mathrm{Hyb}}\left[\mathsf{WIN3.2}_{\mathcal{A}} \cap \mathsf{Fail}_{\mathrm{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right]}_{\leq \mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK}_2,\mathcal{B}_3}}.\end{aligned}$$

It is straightforward to obtain an adversary $\mathcal{B}_3$ (based on $\mathcal{A}$) which has an advantage $\mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{B}_3} = \Pr_{\mathrm{Hyb}}\left[\mathsf{WIN3.2}_{\mathcal{A}} \cap \mathsf{Fail}_{\mathrm{ext}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}} \cup \mathsf{PolicyForge}_{\mathcal{A}}}\right]$. The case for a soundness attack w.r.t. NIZK_1 is analyzed accordingly for event $\mathsf{WIN3.1}_{\mathcal{A}}$ therefore the advantage of $\mathsf{Adv}^{\mathrm{Ext}}_{\mathsf{NIZK},\mathcal{B}_2}$ in the bound follows. In fact, the adversary $\mathcal{B}_3$ 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 $(\mathsf{tok}^* := (F^*, \mathsf{sk}_{F^*}, \sigma^*_{\mathsf{tok}}), \mathsf{pk}^*_S := (\mathsf{vk}^*_{\mathsf{P},S}, P^*_S, \sigma^*_{\mathsf{pub},S}), \mathsf{pk}^*_R := (\mathsf{vk}^*_{\mathsf{P},R}, P^*_R, \sigma^*_{\mathsf{pub},R}), m^*, \sigma^* := (\pi^* := (\pi', \pi''), \sigma'))$, $\mathcal{B}_3$ behaves as $\mathcal{B}_1$ but does not execute the final steps running the extractor, but instead just outputs $(x := (\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_{F^*}, \mathsf{com}_{S,R}), \pi'')$ in case the conditions of $\mathsf{WIN3.2}$ are satisfied (note that the extractor is run as part of the experiment in Definition 2.6). As above, the emulation toward $\mathcal{A}$ is perfect until the point $\mathcal{B}_3$ 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 $\mathsf{WIN3.2}$) but the extraction produces a witness w such that $(x', w') \notin R^2_{\mathrm{ZK}}$. The statement follows. □

□

Attribute-Hiding

Next, we show that our scheme achieves attribute-hiding.

Theorem 5.10. *Let* Π *be an adaptively UC-secure two-party computation protocol for the circuit specified in Figure 16,* $\mathsf{com} = (\mathsf{Setup}, \mathsf{Com})$ *an equivocal commitment scheme,* $\mathsf{PE} = (\mathsf{PE.Setup}, \mathsf{PE.KeyGen}, \mathsf{PE.Enc}, \mathsf{PE.Dec})$ *be a predicate encryption scheme,* $\mathsf{NIZK}_1 = (\mathsf{NIZK}_1.\mathsf{Setup}, \mathsf{NIZK}_1.\mathsf{Prove}, \mathsf{NIZK}_1.\mathsf{Verify})$ *and* $\mathsf{NIZK}_1 = (\mathsf{NIZK}_2.\mathsf{Setup}, \mathsf{NIZK}_2.\mathsf{Prove}, \mathsf{NIZK}_2.\mathsf{Verify})$ *NIZK proof systems (for the relation* R^1_{ZK} *of Figure 17 and for the relation* R^2_{ZK} *of Figure 18, respectively) and* $\mathsf{DS}_{\mathsf{pub}} = (\mathsf{DS}_{\mathsf{pub}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{pub}}.\mathsf{Verify})$ *and* $\mathsf{DS}_{\mathsf{tok}} = (\mathsf{DS}_{\mathsf{tok}}.\mathsf{Setup}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Sign}, \mathsf{DS}_{\mathsf{tok}}.\mathsf{Verify})$ *unforgeable signature schemes, then the construction* $\mathsf{UPCS} = (\mathsf{Setup}, \mathsf{PolUpd}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figures 14 and 15, is attribute hiding.*

Proof. The proof of this theorem proceeds very similar to the proof of the previous interactive UPCS scheme with the difference that we additionally need to rely on the security of the second non-interactive zero-knowledge proof, the equivocality of the commitment scheme and the predicate encryption scheme. We describe the corresponding hybrids in more detail:

Hybrid H_0: This hybrid is defined as the attribute-hiding game for the case that $b = 0$.

Hybrid H_1: This hybrid is the same as in the previous proof. We recap it here for easier readability. In this hybrid, we answer the SIGN query asked by the adversary differently. We denote the modified sign query as SIGN′. A SIGN′ query is defined as a SIGN query with the difference that it is only answered if the queried receiver key has been previously output as an answer to a Test-KeyGen query and the token has been previously output as an answer to a Update query, i.e., for a query $(\textsc{Sign}, \mathsf{tok}_e, \mathsf{pk}_i, m)$ to P_j the public pk_i has been the answer to a query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ and the token tok_e has been the answer to a query $(\textsc{Update}, F_e)$. If this is not the case, then the reply ot the $\textsc{Sign}'$ query is $\perp$. The indistinguishability of H_0 and H_1 is justified by the same reasoning as we have seen for the non-interactive scheme (Figure 8). Namely, by relying on the Lemma 4.4 and Lemma 4.7, as in the proof of Theorem 5.1. For the details of this transition, we refer to the proof of the previous scheme, i.e. Lemma 5.4.

Hybrid H_2: This hybrid is obtained by invoking the security of the 2PC and switch to an ideal execution with functionality $\mathcal{F}_C$. In this hybrid, we change the way the output of every $\textsc{Sign}'$ query that involves party P_j is computed as in the previous section. The transition from H_1 to H_2 follows, as stated previously, from the proof of Lemma 5.5.

Hybrid H_3: This hybrid is almost the same as in the previous proof. We recap and adapt it here for easier readability. In this hybrid, we switch from an honestly generated $\mathsf{CRS}_{\mathsf{NIZK},1}$ and honestly generated proofs in the singing queries to a simulated $\mathsf{CRS}_{\mathsf{NIZK},1}$ and simulated proofs for the output of the two-party computation protocols. That is, upon a PCS signing query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ to P_j, we find the attributes that have been used to generate the key pk_i for P_i, i.e., we find the query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ that was answered using pk_i, as well as the policy F_e that corresponds to the token tok_e, i.e., the query $(\textsc{Update}, F_e)$ which has been answered using tok_e. Further, we also obtain the attributes that are associated with party P_j, i.e., we search for the corresponding query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$. In the next step, we check that $F_e(x_{j,0}, x_{i,0}) = 1$. If this is the case, then we simulate the proof π' that is generated inside the two-party protocol using the NIZK simulator on input the trapdoor and $(\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}_{\mathsf{P}}^{S}, \mathsf{vk}_{\mathsf{P}}^{R}, \mathsf{com}_{S,R})$ where $\mathsf{vk}_{\mathsf{P}}^{S}$ and $\mathsf{vk}_{\mathsf{P}}^{R}$ are part of the public keys of the sender (P_i or P_j) and receiver (P_j or P_i) and the commitment $\mathsf{com}_{S,R}$ that has also been generated during the execution of the two-party protocol. The simulated proof π' is then used as part of the output of the protocol. In any other case (in particular associated attributes do not satisfy the policy), we output $\perp$. As in the previous proof, the transition from H_2 to H_3 is justified by the zero-knowledge property of NIZK_1. For the details of this transition, we refer to the previous proof, i.e., Lemma 5.6.

Hybrid H_4: In this hybrid, we also simulate the proofs of the second proof system NIZK_2. In more detail, we change from an honestly generated $\mathsf{CRS}_{\mathsf{NIZK},2}$ and honestly generated proofs in the final signing step to a simulated $\mathsf{CRS}_{\mathsf{NIZK},2}$ and simulated proofs. That is, upon a PCS signing query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ to P_j, we find the attributes that have been used to generate the key pk_i for P_i, i.e., we find the query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$ that was answered using pk_i, as well as the policy F_e that corresponds to the token tok_e, i.e., the query $(\textsc{Update}, F_e)$ which has been answered using tok_e. Further, we also obtain the attributes that are associated with party P_j, i.e., we search for the corresponding query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$. For the execution of the two-party computation protocol, we behave as in the previous hybrid, i.e., we simulate the proof π that results from the interaction of the two parties. Afterwards, for the generation of the final proof π'' by P_j, we check that $\mathsf{PE.Dec}(\mathsf{sk}_{F_e}, \mathsf{ct}_{S,R}) = 1$, where $\mathsf{ct}_{S,R}$ has been output by the two-party computation protocol and sk_{F_e} is part of tok_e. If this is the case, then the simulator of NIZK_2 is used. In more detail, on input $(\mathsf{sk}_{F_e}, \mathsf{com}_{S,R})$, where $\mathsf{com}_{S,R}$ is also obtained as an output of the two-party computation protocol, the trapdoor is used to simulate the proof π'' which is then used to compute the final signature. The indistinguishability between this and the previous hybrid follows from the zero-knowledge property of NIZK_2, which we formally prove in Lemma 5.11.

Hybrid H_5: In this hybrid, we rely on the equivocator for the generation of the commitment $\mathsf{com}_{S,R}$, which is output during the signing procedure. In more detail, the CRS $\mathsf{CRS}_{\mathsf{Eq}}$ is generated by the algorithm Eq_1 instead of the setup procedure Setup and if a query $(\textsc{Sign}, \mathsf{tok}_e, \mathsf{pk}_i, m)$ is issued for the party P_j, then we generate the commitment $\mathsf{com}_{S,R}$ by executing Eq_2, i.e., $\mathsf{com}_{S,R} \leftarrow \mathsf{Eq}_2(\mathsf{CRS}_{\mathsf{Com}})$, and the corresponding opening r_{com} by executing Eq_3, i.e., $r_{\mathsf{com}} \leftarrow \mathsf{Eq}_3(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R})$ with $\mathsf{ct}_{S,R} \leftarrow \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_{j,0} \dot\cup x_{i,0}))$. The indistinguishability between this and the previous hybrid follows from the equivocality of the commitment scheme, which we formally prove in Lemma 5.12.

Hybrid H_6: In this hybrid, we change the generation of the ciphertext $\mathsf{ct}_{S,R}$ inside the two party computation protocol for a signing query issued for a corrupted P_j acting as the sender. In more detail, if a query $(\textsc{Sign}, \mathsf{tok}_e, \mathsf{pk}_i, m)$ is issued where the key pk_i for party P_i has been generated using a query $(\textsc{Test-KeyGen}, P_i, (x_{i,0}, x_{i,1}))$, we distinguish between two cases: first, the sender is honest and gets corrupted later and, second, the sender is already corrupted. In the second case, we generate the commitment and its opening using Eq_2 and Eq_3 as described in the previous hybrid where $\mathsf{ct}_{S,R} \leftarrow \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_{j,1} \dot\cup x_{i,1}))$ instead of $\mathsf{ct}_{S,R} \leftarrow \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_{j,0} \dot\cup x_{i,0}))$. In the second case, we generate the commitment $\mathsf{com}_{S,R}$ that is used as an output of the signing query by executing $\mathsf{Eq}_2(\mathsf{CRS})$ and, once the sender P_j is being corrupted, we generate the corresponding opening by executing $\mathsf{Eq}_3(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R})$ with $\mathsf{ct}_{S,R} \leftarrow \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_{j,1} \dot\cup x_{i,1}))$. The indistinguishability between this and the previous hybrid can be argued by relying on the attribute-hiding property of the PE scheme, which we formally prove in Lemma 5.13.

Hybrid H_7: In this hybrid, we answer the $\textsc{Test-KeyGen}$ queries using the attributes x_1 instead of x_0. As already mentioned in the end of the previous transition, at this point, the generated signatures are completely independent of the attributes associated with the key. Furthermore, a valid adversary can only ask corruption queries for a key where $x_0 = x_1$. Taking these two facts into account directly results in the computational indistinguishability of the hybrids H_6 and H_7.

Hybrid H_7: In this hybrid, we change the generation of the commitment $\mathsf{com}_{S,R}$ from equivocal mode back to being an honest commitment to the ciphertext $\mathsf{ct}_{S,R}$. In more detail, for a signing query $(\textsc{Sign}, \mathsf{tok}_e, \mathsf{pk}_i, m)$ issued for the party P_j, we generate the commitment $\mathsf{com}_{S,R}$ by executing $\mathsf{Com}(\mathsf{ct}_{S,R}; r_{\mathsf{com}})$ with $\mathsf{ct}_{S,R} \leftarrow \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_{j,1} \dot\cup x_{i,1}))$ where $r_{\mathsf{com}} \leftarrow \{0,1\}^{\lambda}$, instead of $\mathsf{com}_{S,R} \coloneqq \mathsf{Eq}_2(\mathsf{CRS}_{\mathsf{Com}})$.[11] The indistinguishability between this and the previous hybrid follows analogously to the transition from H_4 to H_5. Therefore, we refer to the proof of Lemma 5.12 for further details.

Hybrid H_8: In this hybrid, we change back from a simulated $\mathsf{CRS}_{\mathsf{NIZK},1}$ and the simulated proofs π of the proof system NIZK_1 in the two-party computation protocol back to an honestly generated $\mathsf{CRS}_{\mathsf{NIZK},1}$ and honestly generated proofs π. Since the attributes of the parties have been changed in hybrid H_6 and, therefore, honest statements are proven again, this transition follows symmetrically to the transition from H_3 to H_4.

Hybrid H_9: In this hybrid, we change back from a simulated $\mathsf{CRS}_{\mathsf{NIZK},2}$ and the final simulated proofs π of the proof system NIZK_2 in the signature back to an honestly generated $\mathsf{CRS}_{\mathsf{NIZK},2}$

[11]Here, we also need to change the generation of the CRS from equivocal mode $\mathsf{CRS}_{\mathsf{Eq}}$ back to an honest generation using Setup.

and honestly generated final signature proofs π. Since the attributes of the parties have been changed in hybrid H_6 and, therefore, honest statements are proven again, this transition follows symmetrically to the transition from H_2 to H_3.

Hybrid H_{10}: In this hybrid, we change from a simulated execution of the two-party protocol to an honest execution of the protocol. Symmetrical to the transition from H_1 to H_2, this transition can be argued using the security of the two-party computation protocol Π.

Hybrid H_{11}: This hybrid corresponds to the attribute-hiding game using $b = 1$ as its input. In this game, we change the behavior of the signing queries back from $\textsc{Sign}'$ to $\textsc{Sign}$. Symmetrical to the transition from H_0 to H_1, this transition can be argued using the unforgeability of the public keys and the update tokens.

From the definition of the hybrids, it is clear that H_0 corresponds to the attribute-hiding game with $b = 0$ and H_{11} corresponds to the attribute-hiding game with $b = 1$. Since it holds that $\mathsf{H}_0 \approx_c \cdots \approx_c \mathsf{H}_{11}$, the theorem follows. □

Lemma 5.11 (Indistinguishability of H_3 to H_4)**.** *Let NIZK_2 be non-interactive zero-knowledge proof, then the hybrids H_3 and H_4 are computationally indistinguishable.*

Proof. To show that the hybrids H_3 and H_4 are computationally indistinguishable, we build a reduction to the zero-knowledge property of the NIZK_2 to argue the independence of the final signatures from the commitment, the ciphertexts and, therefore, the attributes of the parties. Before we analyze this case, we stress that all the other queries, i.e., policy update queries, corruption queries and key generation queries, are answered as in the previous hybrid. The only difference here is that the $\mathsf{CRS}_{\mathsf{NIZK},2}$ is obtained from the underlying NIZK_2 challenger and not generated by the reduction.

For a singing query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ asked to party P_j, we can now rely on $\mathcal{F}_C$. In more detail, in the ideal execution, the inputs of both P_j and P_i in the 2PC must be provided to the functionality (either by the environment/reduction or the adversary for corrupted parties). Let $(\mathsf{pk}_j, \mathsf{sk}_i := (\mathsf{vk}^R_{\mathsf{P},i}, \cdot, x_{i,0}, \sigma^i_{\mathsf{priv}}), \mathsf{mpk})$ denote the input of party P_i and $(\mathsf{pk}_i, \mathsf{sk}_j := (\mathsf{vk}^S_{\mathsf{P},i}, \cdot, x_{j,0}, \sigma^j_{\mathsf{priv}}), \mathsf{mpk})$ denote the input of party P_j. We can now evaluate the circuit as per specification $C_{\mathsf{CRS}_{\mathsf{NIZK},1}, \mathsf{mpk}_{\mathsf{PE}}, \mathsf{vk}_{\mathsf{priv}}}$ using the inputs of the parties P_j and P_i and, if all checks of the circuit succeed, we generate the ciphertext $\mathsf{ct}_{S,R} := \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_{j,0} \dot{\cup} x_{i,0}); r_{\mathsf{Enc}})$ with $r_{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$ and the corresponding commitment $\mathsf{com}_{S,R} := \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, \mathsf{ct}_{S,R}; r_{\mathsf{com}})$, with $r_{\mathsf{com}} \leftarrow \{0,1\}^\lambda$. Afterwards, the proof π' is simualted using $(\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P},j}, \mathsf{vk}^R_{\mathsf{P},i}, \mathsf{com}_{S,R})$. Finally, P_j obtains $(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R}, r_{\mathsf{com}}, \pi')$ as an output from the protocol. If any of the previous checks failed, we output $\perp$ to P_j and the signing procedure terminates.

In case the signing procedure succeeds and if $\mathsf{PE.Dec}(\mathsf{sk}_{F_e}, \mathsf{ct}_{S,R}) = 1$, P_j generates the final signature by forwarding $((\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_{F_e}, \mathsf{com}_{S,R}), (\mathsf{ct}_{S,R}, r_{\mathsf{com}}))$ to the prove oracle of the underlying challenger for the NIZK protocol which replies with the proof π'', otherwise it outputs $\perp$. Afterwards, P_j generates the final signature by computing $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}^j_{\mathsf{P}}, (m, \mathsf{pk}_i, \mathsf{com}_{S,R}, \pi := (\pi', \pi'')))$ using the signing key $\mathsf{sk}^j_{\mathsf{P}}$ of party P_j and outputs $(m, \mathsf{pk}_i, \sigma := (\mathsf{com}_{S,R}, \pi, \sigma'))$.

We observe that it follows from the zero-knowledge property that this emulation is perfect, and if the underlying challenger generates the proofs using the witness, then the hybrid H_3 is simulated and if the underlying challenger simulates the proofs, then the hybrid H_4 is simulated. This concludes the proof of the lemma. □

Lemma 5.12 (Indistinguishability of H_4 to H_5)**.** *Let com be an equivocal commitment scheme, then the hybrids H_4 and H_5 are computationally indistinguishable.*

Proof. To show that the hybrids H_4 and H_5 are computationally indistinguishable, we build a reduction to the equivocality of the commitment scheme com to argue the independence of the final signatures from the ciphertexts and therefore the attributes of the parties. Before we analyze this case, we stress that all the other queries, i.e. policy update queries, corruption queries and key generation queries, are answered as in the previous hybrid with the exception that the CRS $\mathsf{CRS}_{\mathsf{com}}$ is obtained from the underlying challenger of the commitment scheme and not generated by the reduction.

For a singing query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ asked to party P_j, we proceed as in the previous hybrid until the generation of the commitment $\mathsf{com}_{S,R}$. To generate the commitment $\mathsf{com}_{S,R}$ for a query to $\mathcal{F}_C$, where P_j uses as an input $(\mathsf{pk}_i, \mathsf{sk}_j := (\mathsf{vk}^S_{\mathsf{P},j}, \cdot, x_{j,0}, \sigma^j_{\mathsf{priv}}), \mathsf{mpk})$ and P_i uses $(\mathsf{pk}_j, \mathsf{sk}_i := (\mathsf{vk}^R_{\mathsf{P},i}, \cdot, x_{i,0}, \sigma^i_{\mathsf{priv}}), \mathsf{mpk})$ as an input, we generate the commitment $\mathsf{com}_{S,R}$ by submitting $\mathsf{ct}_{S,R} := \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, (x_{j,0} \dot{\cup} x_{i,0}))$ to the underlying challenger, which will reply with $(\mathsf{com}_{S,R}, r_{\mathsf{com}})$. Afterwards, the proof π' is simulated and the values $(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R}, r_{\mathsf{com}}, \pi')$ are used as the output of $\mathcal{F}_C$ to P_j. To generate the final signature, we first check that the policy evaluation under both of the possible attributes is equal, i.e., $F_e(x_{j,0}, x_i) = F_e(x_{j,0}, x_i) = 1$ where F_e is the policy associated with the token tok_e. If this check succeeds, then we simulate the proof π'', i.e., the proof for the statement $(\mathsf{sk}_{F_e}, \mathsf{com}_{S,R})$. This proof is then used to generate the final signature, i.e., $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}(\mathsf{sk}^j_{\mathsf{P}}, (m, \mathsf{pk}_i, \mathsf{com}_{S,R}, \pi := (\pi', \pi'')))$ where $\mathsf{sk}^j_{\mathsf{P}}$ is part of the secret key of P_j. Finally, the signature $(m, \mathsf{pk}_i, \sigma := (\mathsf{com}_{S,R}, \pi, \sigma'))$ is output.

We conclude the proof by observing that if the underlying challenger honestly commits to $\mathsf{ct}_{S,R}$, then the hybrid H_5 is simulated and if the underlying challenger generates the commitment $\mathsf{com}_{S,R}$ using Eq_2, then the hybrid H_6 is simulated. $\square$

Lemma 5.13 (Indistinguishability of H_5 to H_6)**.** *Let* PE *be an attribute-hiding predicate encryption scheme, then the hybrids* H_5 *and* H_6 *are computationally indistinguishable.*

Proof. To show that the hybrids H_5 and H_6 are computationally indistinguishable, we build a reduction to the attribute-hiding property of the PE scheme. Before we analyze this case, we stress that all the other queries, i.e., policy update queries and key generation queries, are answered as in the previous hybrid. The only difference here is that the $\mathsf{mpk}_{\mathsf{PE}}$ is obtained from the underlying PE challenger and not generated by the reduction.

For a singing query $(\textsc{Sign}', \mathsf{tok}_e, \mathsf{pk}_i, m)$ asked to party P_j, we proceed as in the previous hybrid, until the generation of the predicate encryption ciphertext $\mathsf{ct}_{S,R}$. Here, we distinguish between two cases, first, the case in which the party P_j is honest and, second, the case in which the party P_j is corrupted. We proceed in the two different cases as follows:

P_j **is honest:** For a query to $\mathcal{F}_C$, where P_j uses as an input $(\mathsf{pk}_i, \mathsf{sk}_j := (\mathsf{vk}^S_{\mathsf{P},j}, \cdot, x_{j,0}, \sigma^j_{\mathsf{priv}}), \mathsf{mpk})$ and P_i uses $(\mathsf{pk}_j, \mathsf{sk}_i := (\mathsf{vk}^R_{\mathsf{P},i}, \cdot, x_{i,0}, \sigma^i_{\mathsf{priv}}), \mathsf{mpk})$ as an input, we proceed as in the previous hybrid without generating the ciphertext and obtaining the commitment $\mathsf{com}_{S,R}$ by executing $\mathsf{Eq}_2(\mathsf{CRS}_{\mathsf{com}})$. Afterwards, the proofs π' and π'' are simulated using $(\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P},j}, \mathsf{vk}^R_{\mathsf{P},i}, \mathsf{com}_{S,R})$ and $(\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_{F_e}, \mathsf{com}_{S,R})$ respectively and the final signature is generated by computing $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}^j_{\mathsf{P}}, (m, \mathsf{pk}_i, \mathsf{com}_{S,R}, \pi := (\pi', \pi'')))$ using the signing key $\mathsf{sk}^j_{\mathsf{P}}$ of party P_j. Afterwards, $(m, \mathsf{pk}_i, \sigma := (\mathsf{com}_{S,R}, \pi, \sigma'))$ is output.

P_j **is corrupted:** For a query to $\mathcal{F}_C$, where P_j uses as an input $(\mathsf{pk}_i, \mathsf{sk}_j := (\mathsf{vk}^S_{\mathsf{P},j}, \cdot, x_{j,0}, \sigma^j_{\mathsf{priv}}), \mathsf{mpk})$ and P_i uses $(\mathsf{pk}_j, \mathsf{sk}_i := (\mathsf{vk}^R_{\mathsf{P},i}, \cdot, x_{i,0}, \sigma^i_{\mathsf{priv}}), \mathsf{mpk})$ as an input, we proceed as in the previous hybrid until the generation of the ciphertext $\mathsf{ct}_{S,R}$. To obtain the ciphertext, the values $((x_{j,0} \cup x_{i,0}), (x_{j,1} \cup x_{i,1}))$ are submitted to the underlying challenger which replies with $\mathsf{ct}_{S,R}$.

To generate the commitment $\mathsf{com}_{S,R}$, the two equivocators are executed, i.e., $\mathsf{com}_{S,R} \leftarrow \mathsf{Eq}_2(\mathsf{CRS}_{\mathsf{com}})$ and $r_{\mathsf{com}} \leftarrow \mathsf{Eq}_3(\mathsf{com}_{S,R}, \mathsf{ct}_{S,R})$. Afterwards, the proofs π' and π'' are simulated using $(\mathsf{CRS}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{priv}}, \mathsf{vk}^S_{\mathsf{P},j}, \mathsf{vk}^R_{\mathsf{P},i}, \mathsf{com}_{S,R})$ and $(\mathsf{CRS}_{\mathsf{com}}, \mathsf{sk}_{F_e}, \mathsf{com}_{S,R})$ respectively and the final signature is generated by computing $\sigma' \leftarrow \mathsf{DS}_{\mathsf{P}}.\mathsf{Sign}(\mathsf{sk}^j_{\mathsf{P}}, (m, \mathsf{pk}_i, \mathsf{com}_{S,R}, \pi := (\pi', \pi'')))$ using the signing key $\mathsf{sk}^j_{\mathsf{P}}$ of party P_j. Afterwards, $(m, \mathsf{pk}_i, \sigma := (\mathsf{com}_{S,R}, \pi, \sigma'))$ is output.

If a corruption query is asked for a party P_j for which signatures have previously been generated for the parties $P_{i_1}, \dots, P_{i_k}$, then let $\mathsf{com}^1_{S,R}, \dots, \mathsf{com}^k_{S,R}$ be the corresponding commitments. Since those commitments have been generated using Eq_2, no openings $r^1_{\mathsf{com}}, \dots, r^k_{\mathsf{com}}$ are known yet. To generate these openings, the values $((x_{j,0} \cup x_{i_1,0}), (x_{j,1} \cup x_{i_1,1})), \dots, ((x_{j,0} \cup x_{i_k,0}), (x_{j,1} \cup x_{i_k,1}))$ are submitted to the underlying challenger to obtain the ciphertexts $\mathsf{ct}^1_{S,R}, \dots, \mathsf{ct}^k_{S,R}$. Afterwards the equivocator Eq_3 is executed, i.e., $r^1_{\mathsf{com}} \leftarrow \mathsf{Eq}_3(\mathsf{com}^1_{S,R}, \mathsf{ct}^1_{S,R}), \dots, r^k_{\mathsf{com}} \leftarrow \mathsf{Eq}_3(\mathsf{com}^k_{S,R}, \mathsf{ct}^k_{S,R})$. These values are then used together with the proofs $\pi'_1, \dots, \pi'_k$ generated during the signing process as an output to the adversary.

To analyze this simulation, we observe that if the underlying challenger encrypts $(x_{j,0} \dot\cup x_{i,0})$ for the generation of the ciphertext $\mathsf{ct}_{S,R}$, then the hybrid H_5 is simulated and if the underlying challenger uses $(x_{j,1} \dot\cup x_{i,1})$ for the generation of the ciphertext $\mathsf{ct}_{S,R}$, then the hybrid H_6 is simulated.

To conclude the proof, we need to rely on the perfect correctness of the encryption scheme and the validity of the adversary against the UPCS scheme to argue that no decryption errors allow an adversary to distinguish between two hybrids.

If the adversary towards UPCS is valid, then it means that it only requests signatures and updates, such that it holds that $F_e(x_{j,0}, x_{i,0}) = F_e(x_{j,1}, x_{i,1})$ and, furthermore, that for all corrupted parties P_j, it holds that $F_e(x_j, x_{i,0}) = F_e(x_j, x_{i,1})$ with $x_j := x_{j,0} = x_{j,1}$ for all policy updates F_e. This, in turn, implies that all the ciphertexts, that are generated by the underlying challenger, are generated using challenges $((x_{j,0} \dot\cup x_{i,0}), (x_{j,0} \dot\cup x_{i,0}))$ for which it holds that $F_e(x_{j,0}, x_{i,0}) = F_e(x_{j,1}, x_{i,1})$ where F_e are all the submitted policies for update queries. Taking into account the perfect correctness of the predicate encryption scheme, it follows that $\mathsf{PE}.\mathsf{Dec}(\mathsf{sk}_{F_e}, \mathsf{ct}_{S,R}) = F_e(x_{j,0}, x_{i,0})$ and $\mathsf{PE}.\mathsf{Dec}(\mathsf{sk}_{F_e}, \mathsf{ct}'_{S,R}) = F_e(x_{j,1}, x_{i,1})$ with probability 1. This mirrors the correct policy evaluation in both cases and, therefore, an adversary is not able to distinguish between the two hybrids. ☐

Acknowledgments

The second author was partly supported by the European Union project (ERC AdG REWORC - 101054911), and also by True Data 8 (Distributed Ledger & Multiparty Computation) under the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. The fourth author was supported by an MC2 postdoctoral fellowship.

References

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015.

[AJ17] Prabhanjan Ananth and Abhishek Jain. On secure two-party computation in three rounds. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 612–644. Springer, Heidelberg, November 2017.

[ARYY23] Shweta Agrawal, Mélissa Rossi, Anshu Yadav, and Shota Yamada. Constant input attribute based (and predicate) encryption from evasive and tensor lwe. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 532–564, Cham, 2023. Springer Nature Switzerland.

[ATY23] Shweta Agrawal, Junichi Tomida, and Anshu Yadav. Attribute-based multi-input fe (and more) for attribute-weighted sums. In *Annual International Cryptology Conference*, pages 464–497. Springer, 2023.

[AYY22] Shweta Agrawal, Anshu Yadav, and Shota Yamada. Multi-input attribute based encryption and predicate encryption. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 590–621. Springer, Heidelberg, August 2022.

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017.

[BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 520–537. Springer, Heidelberg, March 2014.

[BGG$^+$90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, Heidelberg, August 1990.

[BKS16] Zvika Brakerski, Ilan Komargodski, and Gil Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 852–880. Springer, Heidelberg, May 2016.

[BMW21] Christian Badertscher, Christian Matt, and Hendrik Waldner. Policy-compliant signatures. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 350–381. Springer, Heidelberg, November 2021.

[BS04] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 168–177. ACM Press, October 2004.

[BSW23] Christian Badertscher, Mahdi Sedaghat, and Hendrik Waldner. Unlinkable policy-compliant signatures for compliant and decentralized anonymous payments. Cryptology ePrint Archive, Paper 2023/1070, 2023. https://eprint.iacr.org/2023/1070.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDLQ16] Hui Cui, Robert H. Deng, Yingjiu Li, and Baodong Qin. Server-aided revocable attribute-based encryption. In Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows, editors, *ESORICS 2016, Part II*, volume 9879 of *LNCS*, pages 570–587. Springer, Heidelberg, September 2016.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, Heidelberg, April 2012.

[CM21] Leixiao Cheng and Fei Meng. Server-aided revocable attribute-based encryption revised: Multi-user setting and fully secure. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021, Part II*, volume 12973 of *LNCS*, pages 192–212. Springer, Heidelberg, October 2021.

[FFMV23] Danilo Francati, Daniele Friolo, Giulio Malavolta, and Daniele Venturi. Multi-key and multi-input predicate encryption from learning with errors. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, pages 573–604. Springer, 2023.

[For87] Lance Fortnow. The complexity of perfect zero-knowledge (extended abstract). In Alfred Aho, editor, *19th ACM STOC*, pages 204–209. ACM Press, May 1987.

[GGG$^+$14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th FOCS*, pages 612–621. IEEE Computer Society Press, October 2017.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Heidelberg, August 2015.

[ISE+18] Ai Ishida, Yusuke Sakai, Keita Emura, Goichiro Hanaoka, and Keisuke Tanaka. Fully anonymous group signature with verifier-local revocation. In *Security and Cryptography for Networks: 11th International Conference, SCN 2018, Amalfi, Italy, September 5–7, 2018, Proceedings 11*, pages 23–42. Springer, 2018.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[KNT18] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 603–648. Springer, Heidelberg, April / May 2018.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162. Springer, Heidelberg, April 2008.

[Lin10] Yehuda Lindell. Foundations of cryptography 89-856. http://u.cs.biu.ac.il/~lindell/89-856/complete-89-856.pdf, 2010.

[LLNW14] Adeline Langlois, San Ling, Khoa Nguyen, and Huaxiong Wang. Lattice-based group signature scheme with verifier-local revocation. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 345–361. Springer, Heidelberg, March 2014.

[LNWZ19] San Ling, Khoa Nguyen, Huaxiong Wang, and Juanyang Zhang. Server-aided revocable predicate encryption: formalization and lattice-based instantiation. *The Computer Journal*, 62(12):1849–1862, 2019.

[MPR11] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 376–392. Springer, Heidelberg, February 2011.

[NWZ16] Khoa Nguyen, Huaxiong Wang, and Juanyang Zhang. Server-aided revocable identity-based encryption from lattices. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16*, volume 10052 of *LNCS*, pages 107–123. Springer, Heidelberg, November 2016.

[OT12] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 591–608. Springer, Heidelberg, April 2012.

[RPB+19] Théo Ryffel, David Pointcheval, Francis Bach, Edouard Dufour-Sans, and Romain Gay. Partially encrypted deep learning using functional encryption. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th FOCS*, pages 600–611. IEEE Computer Society Press, October 2017.