Unlinkable Policy-Compliant Signatures for Compliant and Decentralized Anonymous Payments

Christian Badertscher[1], Mahdi Sedaghat[2], and Hendrik Waldner[3,4]

[1] Input Output, Zürich, Switzerland
christian.badertscher@iohk.io
[2] COSIC, KU Leuven, Leuven, Belgium
ssedagha@esat.kuleuven.be
[3] University of Maryland, College Park, US
hwaldner@umd.edu
[4] Max Planck Institute for Security and Privacy, Bochum, Germany

Abstract. Privacy-preserving payment systems face the difficult task of balancing privacy and accountability: on one hand, users should be able to transact privately and anonymously, on the other hand, no illegal activities should be tolerated. The challenging question of finding the right balance lies at the core of the research on *accountable privacy* that stipulates the use of cryptographic techniques for policy enforcement. Current state-of-the-art systems are only able to enforce rather limited policies, such as spending or transaction limits, or assertions about single participants, but are unable to enforce more complex policies that for example jointly evaluate both, the private credentials of sender and recipient, such as admissible cross-border payments, let alone to do this without auditors in the loop during payment. This severely limits the cases where decentralized virtual assets can be used in accordance with regulatory compliance such as the Financial Action Task Force (FATF) travel rule, while further retaining strong privacy features.

We present *unlinkable Policy-Compliant Signatures (ul-PCS)*, an enhanced cryptographic primitive extending the work of Badertscher et al. (TCC 21). We give rigorous definitions, formally proven constructions, and benchmarks using our prototype developed using CharmCrypto. Unlinkable PCS has the following unique combination of features:

1. It is an enhanced signature scheme where the public key encodes in a privacy-preserving way the user's verifiable credentials (obtained from a credential authority).
2. Signatures can be created (and later publicly verified) by additionally specifying a recipient's public key aside of the to-be-signed message. A valid signature can only ever be created if the attributes x_S of the signer and the attributes x_R of the receiver fulfill some global policy $F(x_S, x_R)$.
3. The signature can be created by the signer just knowing the recipient's public key; there is no further interaction needed no attributes are leaked (beyond the validity of the policy).
4. Once credentials are obtained, a user can generate fresh public keys without interacting with the credential authority.

By merging the act of signing a transaction with the act of providing an assurance about the involved participants being compliant with complex policies, yet retain that participants are able to change addresses without the involvement of an authority, we show how ul-PCS constitutes a crucial step towards achieving a technology that improves regulatory compliance of privacy coins such as Monero or Zcash.

1 Introduction

The inception of Bitcoin [53] and its novel approach to implement a transaction ledger via a blockchain brought to light a new type of payment system that does not rely on trusted parties like a central bank, but instead uses distributed ledger technology to settle direct transactions between parties and to protect against double-spends. Besides Bitcoin, decentralized anonymous payment (DAP) systems, such as Zcash [11] and Monero [3], have been proposed to improve privacy and anonymity guarantees. In these systems, parties enjoy full transaction privacy and anonymity, which makes it challenging to hold parties accountable for their actions, let alone for a regulator to be assured regulatory compliance is met. This led to the study of accountability and auditability in the context of distributed payment systems with the main goal of understanding the necessary adoption requirements of these systems in various jurisdictions [24].

The core meaning of auditability is to provide means to a regulator to ask for specific pieces of information, based on a legal system defining a catalogue of admissible questions, and be given the answer in a faithful way [24]. Clearly, an auditor only needs to (reactively) ask for information that the system does not proactively enforce by itself. This policy enforcement is precisely how accountability for private payment systems, or *accountable privacy* for short, has been defined in [24, 40]. However, in many of the currently proposed systems, the granularity of accountable privacy is not very high, and the focus lies on (functions about) the monetary value of transactions or spending limits on accounts [23, 40, 54, 63], where more centralized designs typically allow for a richer set of provable statements. To make matters worse, auditability is often equated explicitly or implicitly with the ability of an auditor to revoke the privacy and anonymity of transactions of any individual user (or given a key to supervise or view the transaction log) [4, 8, 21, 26, 28, 50]. While this trivially avoids the need for more sophisticated policy enforcement techniques, it goes without saying that such a powerful revocation capability is problematic as it could be subject to abuse.

In light of this, an important question arises: how precisely can we enforce policies on the transaction level? A blueprint followed by several works [33, 59, 64] to obtain accountable privacy in DAP systems is to consider transaction types for specific use cases where each use case is governed by a policy whose compliance can be enforced. In exchange for a potentially limited scope, users get back full and unrevocable anonymity for these transactions. For example the UTT system [59] defines a digital analogon of cash: the so-called budget coins are issued in a limited fashion to certified users. As soon as the user has obtained the coins, they can transact in a cryptographically secure way that, among other things, ensures full, unrevocable privacy.

Central to the success of such systems is the level of granularity for which one can enforce policy compliance. The richer the class of cryptographically enforceable policies, the easier it is to define different transaction types. In view of the developing ecosystem on digital credential systems [18, 25, 62], more legal policies can be translated to the digital world, such as predicates about which two individuals are allowed to transact based on age or residency, or on accredited attributes like financial reliability, credit worthiness, or other real-word certifications. More concretely, this leads to more fine-grained transaction types for which full privacy can be guaranteed. An example could be to lift certain limits for, e.g., residents spending coins in shops that are certified to only sell goods for everyday life. If such policies are enforceable by cryptographic means in a DAP system, this would heavily boost the privacy of users while satisfying regulatory needs. Unfortunately, there is no DAP system that supports this functionality and integrates smoothly with a credential issuance infrastructure. One reason is the strong, at first sight contradictory, set of requirements, which are that (1) all credentials of a user must remain private, even during payment; (2) it must be possible to perform policy evaluation jointly on both, sender and receiver, attributes (such as whether they have the same residence, or whether they belong to jurisdictions across which money transfers are permitted); and (3) compatibility with DAP systems must be guaranteed, which means that evaluation must be possible whenever the sender knows just the recipient's public key (and no further interaction is required to submit the transaction) and that compliance is publicly verifiable. We note in passing that even centralized designs such as Platypus [64] do currently not support this stronger type of joint policy evaluation (but admit individual attestations of users about their own attributes toward the central bank).

In this work, we give a generically applicable cryptographic policy-enforcement mechanism that is suitable to be integrated with DAP systems, i.e., it satisfies properties (1)-(3) above. The mechanism is generic—it has the interface of a signature scheme—and can be modularly composed with larger systems to complement existing solutions to achieve fine-grained accountable privacy. However, our solution is not limited to decentralized ledgers, and can also be applied to centralized designs to reduce the information leakage about a transaction towards the auditor.

1.1 Contributions

Definitions. We introduce the enhanced cryptographic primitive called unlinkable policy-compliant signatures, a stronger version of policy-compliant signatures introduced by Badertscher, Matt, and Waldner [6]. We give precise definitions of unforgeability, attribute-hiding, and unlinkability. Since privacy (resp. anonymity) and unforgeability are intertwined in this definition, special care must be taken to arrive at a reasonable definition.

Generic solution. We provide a generic solution to the problem that realizes ul-PCS for arbitrary policies $F(x, y)$ defined for sender and receiver attributes x and y, respectively, and formally prove its security. Despite its seemingly theoretical focus, the main practical challenges in instantiating this primitive are the predicate encryption (PE) scheme and the non-interactive zero-knowledge (NIZK) proof systems. We present an implementation of our generic construction for the inner-product (IP) predicate, i.e., for vectors x and y of attributes (encoded as field elements) such that $F(x, y) = 1$ iff the inner product of x and y is zero. This predicate is known to be sufficient to realize many real-world policies including set membership (e.g., used in identity-based revocation systems), CNF formulas and exact threshold clauses (with conjunctive or disjunctive clauses) as well as hidden-vector encryption enabling various sorts of comparisons as well as conjunctions of the above statements [6].

More efficient constructions and implementations. We provide two additional, specific constructions for certain policy classes that are more lightweight in terms of required cryptographic tools and have the additional features of constant-sized public keys and signatures, as well as constant verification time. We provide prototype implementations in Python and utilize the Charm-crypto framework [2] for all constructions we present in this work, including the generic solution (instantiated with an IP-PE scheme).

For the experiments, we use a PC (laptop) with Intel Core i7-9850H CPU @ 2.60 GHz and 16 GB memory over the BN-254 curves. Our benchmarks demonstrate that the signing execution time for the specific schemes is less than 2 seconds for reasonably complex policy sizes. The verification procedure of signatures takes around 1.6 seconds, with public keys in the order of 28 kbytes, and signature sizes of about 16 kbytes (all independent of the number of attributes). Our prototypes for the specific policy classes thus suggest that, although ul-PCS may at first sight appear like a heavy cryptographic tool, they are able to enforce policies with reasonable practical efficiency.

Finally, our prototype for the generic solution gives a first estimate about the real-world complexity of general-purpose PCS designs. Due to its relationship with predicate-encryption (which we explain in Section 2 below), the performance is largely influenced by the choice of PE scheme. We run our benchmarks in the range of $n = 5$ up to 50 attributes. Signing takes 3 seconds for $n = 5$ and each additional attribute incurs, on average in that range, an estimated cost of 340 ms. For verification, we obtain around 4.5 seconds for $n = 5$ with an average cost per additional attribute of around 420 ms. Public key sizes on the other hand grow linearly, starting at 79 kbytes for $n = 5$, and incurring a cost of roughly 9.9 kbytes per additional attribute (which means that even for 50 attributes, we have key sizes similar to McEliece cryptosystems). Signatures are about 42 kbytes for $n = 5$ and grow by 5.14 kbytes per additional attribute. We compare these characteristics with the suggested PCS construction from [6], for which we provide the first prototype (with the same underlying PE scheme), which enables us to directly observe the overhead that our generic anonymity enhancement techniques impose.

Application to payment systems. We show how to integrate ul-PCS with UTxO ledgers, as well as with DAP systems like Zcash or Monero to ensure fine-grained policies on the transaction level without the involvement of an auditor. To this aim, we build on a recent abstract framework by Engelmann et al. [31] to modularly compose ul-PCS with so-called one-time accounts, effectively coupling addresses with private credentials. We point out that this integration does not introduce any additional trust assumptions beyond what a credential issuance infrastructure would need. We further show that credential issuance, more formally, setup and key-generation, can be distributed across a set of servers to avoid a single point of failure. This is an important consideration, since corrupting the credential issuer usually enables an adversary to generate keys by itself, which it can then use to brute force the attributes of every participant in the system by checking to which participants it is able to send based on its self-issued attributes. Finally, in the centralized settings of CBDC constructions [50, 64], we showcase how an integration of ul-PCS extends the set of policies that could be automatically enforced.

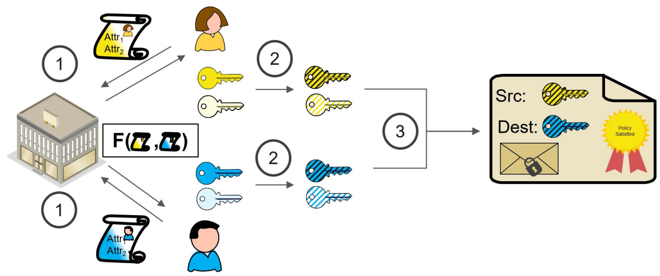

Fig. 1: Usage of an unlinkable policy-compliant signature scheme: 1.) Alice with credentials x and Bob with credentials y, run through a registration process with a credential authority. 2.) At any point, they can decide to re-randomize their keys in order to break any link to their previous actions. 3.) Alice generates a signature, e.g., on a private transaction, with Bob's public key as the destination address. The fact that a valid signature emerges proves that $F(x, y) = 1$.

2 Technical Overview and Related Work

In this section, we give an in-depth overview of the technical contributions of this work. We describe the security goal of ul-PCS and the inherent complications to realize such a strong notion. Afterwards, we outline the constructions we present in this work and the different types of policies that we support. Finally, we provide an overview how DAPs can be enhanced with ul-PCS in the context of FATF regulations [34], finding a good balance between privacy and regulatory-friendliness, as outlined in the previous section. We conclude by an overview of related work.

2.1 Realizing ul-PCS for Various Types of Policies

We begin by recalling the model behind PCS, put forth in [6]. The model involves three main roles as depicted in Figure 1: the Credential Issuing Authority (CA), Signers and Verifiers. The policy can be defined for a set of senders' attributes $\mathcal{S}$ and receivers' attributes $\mathcal{R}$ such that a predicate function $F : \mathcal{S} \times \mathcal{R} \rightarrow \{0, 1\}$ determines which senders, with a given set of attributes, are allowed to create a signature for which receivers, that possess a subset of attributes. If x and y are the (private) attributes of sender and receiver, respectively, then creating a valid signature is allowed iff $F(x, y) = 1$. *Existential unforgeability* demands that a signer cannot generate a valid signature for a recipient, identified by its public key (again with attributes y), unless it has obtained the secret key associated with x (issued by the CA) such that $F(x, y) = 1$. *Attribute hiding* guarantees that nobody learns any meaningful information about the attributes of the signer and targeted verifier except, of course, the bit that they jointly fulfill the policy when a valid signature emerges. We identify and introduce a missing feature: *unlinkability*. A user must be able to change the representation (i.e., its public key) without interacting with the authority, to break the link between its actions—while retaining all security features above. Since ul-PCS combines anonymity with, for example, unforgeability requirements, the existing security games must be adapted. A resulting challenge is that winning conditions must remain well-defined, even if keys are re-randomized (possibly done by the adversary) and attributes are hidden.

Design challenges. At first sight, the problem might appear not too difficult as it (superficially) resembles anonymous credential (AC) systems [19,25], which are well-studied and play a key role in privacy-preserving applications by enabling users to authenticate themselves while ensuring the unlinkability of their actions. A credential in this context is typically a signature on the attributes. During the authentication process, users can demonstrate their possession of a credential that satisfies a specific access policy without revealing any details about the real identity of the user, except that they meet the criteria of the access policy. While one can notice some similarities with our goal set out above, ul-PCS possesses distinctive properties that deviate crucially from the intuition about how AC systems are used.

The first difference is in the representation of the credentials. In ul-PCS the credential is an inherent part of the public information as its intended use case is as an address in a payment system, cf. Section 2.2. In contrast, credential systems typically assume that all attributes remain private unless shown in a credential-show operation. In most implementations, a credential is simply a set of attributes signed by an authority. As a consequence, re-randomization in our context means that we have to find a privacy-preserving representation of users' credentials that is fully re-randomizable without contacting the authority again. This departs from the usual requirements that AC systems must fulfill.

Furthermore, generic credential-show algorithms are not necessarily privacy-preserving, unless they involve assertions proved in zero-knowledge. In this context, these are properties that a user proves about its private attributes. In contrast, in order to fulfill the desired goals of ul-PCS the signer/sender needs to assert a policy that involves the private attributes of itself and the receiver—without having the receiver disclose the information to the sender. Any ul-PCS system must thus satisfy a set of seemingly incompatible requirements which makes the problem highly non-trivial to solve.

Scheme for generic policies. We first consider the case for generic policies that achieve the goal of an arbitrary joint check of the sender's and receiver's attributes encoded (in a privacy-preserving manner) in their public keys. The standard PCS construction in [6] relies on three main cryptographic primitives: (Predicate-Only) Predicate Encryption, Digital Signatures and Non-Interactive Zero-Knowledge (NIZK) proofs which can all be instantiated in the standard model. The high-level idea of this construction is that public keys of parties, acting as receivers, contain a PE-ciphertext decryptable by the signer/sender, only upon policy satisfaction. The signature and NIZK are needed to achieve unforgeability and to prove multiple relations during the signing process. Following this paradigm, we present the first unlinkable PCS scheme supporting any policy in Section 5. We build our scheme using the primitives mentioned above, and embed a method that allows a party to evolve its public key, according to a pseudo-random sequence tied to their attributes. A critical hurdle to overcome in this setting is that, during the process of refreshing the key, a party cannot add new attributes that have not been issued to that party. We present more details on this in the technical sections.

Schemes for more specific policy classes. Despite the fact that PE is the most elegant conceptual fit for general PCS, it impacts efficiency and public-key sizes since there is a direct implication between PCS and PE. In more detail, the reduction presented in [6] shows that PCS gives rise to PE encryption (for a related predicate), and, furthermore, that a PCS public key can be turned into a PE ciphertext. This implication becomes even more dominant in the ul-PCS case since, in this case, it is required, as part of the construction, to prove the well-formedness of the public key. To improve this situation, we show how, for specialized policies, it is possible to avoid PE in order to obtain more practically performing schemes. We consider two specific policy classes:

Scheme for role-based policies. We consider role-based access-control matrices. Such a matrix can be seen as a function $F : [n_R] \times [n_R] \to \{0, 1\}$ (for a given, presumably relatively small, set of n_R number of roles) and captures which roles i can transact towards which role j, namely iff $F(i, j) = 1$. Depending on the structure of such a matrix, one can implement a wide range of access control policies, where "access control" here rather means which role is allowed to send a signed message (or transaction) to which other role akin to information flow in [20]. Of particular interest is the special case of equality, i.e., the identity matrix $F(i, j) = 1$ iff $i = j$ [5] as we recall below. We present an approach based on accumulators (which are realizable based on pairing-based signatures of a specific type) instead of PE. For general RBAC matrices, the scheme satisfies what we call outsider-secure attribute hiding, sometimes referred to as transaction-graph obfuscation or confidentiality [21, 24] (aside of unforgeability and unlinkability). This security notion captures the inability of an attacker to infer any useful information by just analyzing the transaction graph of a blockchain. For the special case of the equality policy, where users are grouped into categories, the scheme satisfies all security properties (in particular full attribute-hiding). The equality policy is particularly useful in contexts where users and/or services are clustered into groups or categories based on their real-world credentials, and to ensure that transactions are conducted within those groups.

Scheme for separable policies. Separable policies are policies that admit the simple representation $F(x, y) = S(x) \wedge R(y)$, and thus belong to the important class of predicates w.r.t. individual assertions about an entity's attributes, e.g., the ones supported by centralized solutions like Platypus [64]. We show that those policies can be realized by unlinkable PCS schemes in an efficient way, where the PE part can be replaced by standard public-key encryption. We point out some of the applications of these policies: on one hand, $S(x)$ could be the predicate that a sender has undergone KYC regulations, while a priori anyone can be a receiver ($R(y) = 1$). Translated to our scheme, and its associated usage in a DAP system, this means that anyone can immediately start off and receive coins, while only being able to spend them later, once KYC regulations have been met. The second, more technical use-case, appears in Zcash-like DAP systems: the three transaction types in Zcash, namely

Mint, Burn, and Pour transactions can directly be lifted to user roles: Mint is an action from a sender-only role ($S(x) = 1, R(y) = 0$), Burn is a transaction toward a receiver-only role ($S(x) = 0, R(y) = 1$), and the normal use-case is a user that can send and receive ($S(x) = R(y) = 1$). Combining this observation with the results of Section 7, gives a generic way to let the monetary policy be governed by accredited users while preserving their privacy and anonymity. In addition, when integrating Zcash with other ledger-based currencies, one can steer which users are allowed to convert base currency in exchange for newly minted zerocoins.

2.2 FATF, DAPs, and ul-PCS

Virtual assets is the technical and legal term referring to decentralized digital tokens that are considered cryptocurrencies. Such virtual assets can either be stored in self-hosted wallets, or stored in a hosted (or custodial) wallet on a virtual-asset exchange, more generally referred to as virtual-asset service provider (VASP) [34]. While digital assets serve real financial and investment needs, to protect the ecosystem from fraudulent and criminal activities, the Financial Action Task Force (FATF) demands that VASPs comply with the so-called travel rule. The travel rule mandates that VASPs maintain identifying information behind any address they store, as well as to collect and exchange information about sender and receiver when funds move from one hosted wallet to another, and further verify that certain (legal) policies are satisfied, such as restrictions on capital flow between jurisdictions of the involved legal entities.

The travel rule puts a lot of burden on VASPs.[5] Identifying the financial beneficiary behind any address is similar to solving the lookup problem in PKI infrastructures: it must be efficiently possible for any VASP, when given an address addr, to obtain the identifying information behind addr, and most importantly, the VASP that is hosting addr (if any). Since these checks are the precursor for sensitive information exchanges about financial individuals, the accuracy of such an association is of utmost importance: a (curious) VASP should not be able to learn such information if it cannot present a proof that it is the custodian of either the sending or receiving wallet, however, it should be efficiently possible to verify whether a wallet is hosted. Realizing such a lookup service as an overlay over decentralized tokens is a difficult endeavor, as personal information is stored, maintained, transferred, and replicated on various VASPs, which is not just a concern related to privacy, but also mandates that information about an individual must be consistent. Even if these issues were resolved, it appears that the FATF travel rule does not align well with anonymous payment systems. This is due to the strong anonymity guarantees that these assets offer, which deems them suspicious, mainly due to a lack of technological capabilities of reconciling accountability and privacy efficiently for decentralized assets.

In this paper, we put forth a mechanism, which we formalize later in Section 7, enabling the reconciliation of the above views, the silver lining being that an address addr already provides an encoding of credentials with it—encoded in a privacy-preserving way via the help of a credential authority issuing any sort of attributes to participants. This achieves

[5] https://sanctionscanner.com/blog/financial-action-task-force-fatf-travel-rule-140

a separation of duties: the identifying information is carried by the address itself, and its privacy features, such as recoverability or privacy revocation features, is up to the credential system, not the VASPs. While the idea of connecting addresses with credentials is not really novel, what ul-PCS adds to the system is the combination of two new features:

– It enables that an address carries anonymous credentials, but also has the look-and-feel of common cryptocurrencies: a user can create fresh addresses by itself that carry the same information, without the need to contact the credential authority.
– Asset transfers can be automatically governed by a policy $F(x, y)$, where x are the attributes of the sender and y are the attributes of the receiver. That is, a transaction transferring a token from addr_x to addr_y can never be accompanied by a valid signature unless $F(x, y) = 1$. Furthermore, nothing more is leaked by such signatures other than the validity of the policy.

These two features combined can improve the complex situation faced by VASPs considerably, while maintaining user privacy. As we define formally in Section 7.2, compliance checks based on ul-PCS and spending rights in a DAP system can be seen as two separate steps, similar to multi-signatures. A VASP can formally host an address by controlling just the DAP private key, while the ul-PCS private key always remains with the user. To conduct an asset transfer, the user and the VASP must both provide the signature. Still, a user can have many different unlinkable addresses with various VASPs thanks to the re-randomization property that allows it to create fresh addresses. Finally, while the above solution works best if the underlying blockchain allows native support of such addresses and multi-signatures, we point out that blockchains with smart-contract capabilities can support these types of operations by defining a custom token controlled by two keys.

Asset transfers implemented this way are guaranteed to follow the policy—without ever requiring from the involved parties to disclose any information about their attributes–and reliefs the VASPs from collecting (or transmitting) information that are made for the sole purpose of checking compliance of the mentioned policy (of course, there might still be a need to collect information that is not formalized by a digital policy in which case ul-PCS helps reducing the amount of collected information). Furthermore, the asset transfer is non-interactive in the same sense as common cryptocurrencies are: the sender does not need more information to transfer the asset than the knowledge of the recipient's address.

It further allows the VASP to outsource the task of KYC to accredited authorities. Here, the authorities issue attributes to reflect a user's KYC status which are then in charge of delivering the associated information, if requested by legal enforcement. We discuss such possibilities in Section 7.4. Furthermore, a VASP itself (such as a mixing service) can carry a ul-PCS key representing attributes accredited to it. This way, a policy can specify what types of users are allowed to use which specific services. Only those users would be able to transfer assets into an account of a VASP that satisfies the rules. We discuss this example in more detail in Section 7.1.

In summary, the proposed approach is a paradigm change in handling accountability in transaction systems. Instead of enforcing an overlay-structure where every VASP collects the same type of information, which arguably is rather intrusive, we put forth a cryptographic mechanism that turns this view upside down. More precisely, we intimately connect the addresses to the credentials with automated compliance checks, while allowing a user to be represented by fresh addresses with different services.

2.3 Prototypes and Benchmarks

We provide a full prototype implementation for all the schemes we present in this work. These are the first working prototypes for policy-compliant signatures in general, for which we provide the results in Section 6. Our prototype should be seen as an academic prototype that contains a first faithful implementation of all building blocks including the zero-knowledge proof systems, however, without production-grade optimizations (we mention a few as open directions). Yet, even without these modifications, the main benefit of the prototypes, besides of obtaining concrete runtime estimates, is the required dovetailing of the zero-knowledge system with other cryptographic primitives. It is highly non-trivial how a concrete instantiation of our generic scheme would actually look like (and to what extent it follows from standard tools). For that instantiation, we pick a predicate-encryption scheme for the class of inner-product (IP) policies [55]. Those schemes are theoretically efficient and inner-product predicates are known to realize various complex policies [49] such as DNF/CNF formulas, threshold clauses, or polynomial evaluations, which directly translate to the PCS setting [6]. Furthermore, since hidden-vector encryption can be realized from IP, it follows that IP is sufficient to implement all policies from [16].

The main challenge to overcome from a practical perspective is to achieve that a user is able to provably re-encrypts the particular PE ciphertexts without introducing new attributes. While this is easy on paper, we must implement this securely by using a combination of structure-preserving signatures on equivalence classes and the observation that the particular PE ciphertexts are closely related to generalized Pedersen commitments for which we can achieve re-randomization via its homomorphic property. We thereby are able to show that we can couple the particular PE scheme, which is based on dual pairing vector spaces, with standard NIZK systems (such as Groth-Sahai and Sigma protocols). The full specification of all our prototypes are given in Appendices C and D.

2.4 Related Work

Since ul-PCS are signatures, they can be composed with any transaction system to capture more fine-grained policies. We already contrasted this paper with [6], which serves as the basis we extend. Therefore, we now focus on an overview of how ul-PCS can improve the expressiveness of existing payment systems. We present the technical details on this later in Section 7. In the context of distributed payment systems, the focus of prior works that support accountable privacy is on using NIZKs to prove statements about the content of a transaction such as, e.g., a sequence of transactions are below a spending limit in total or below a receiving limit in total [40, 59, 63]. These policies are extremely useful and the involved NIZKs are practical. The systems are therefore able to publicly convince an auditor that certain actions are within limits but do not give assertions about the credentials of the involved parties and it seems hard to obtain unrevocable privacy for more than cash-like transactions [59]. Enriched with digital credentials, however, more classes of transactions can be defined. Parties involved in a payment could be accredited (trusted) institutions or shops, for which a sending or receiving limit is lifted. A PCS signature signing the transaction can assure money flow only between two such institutions. Furthermore, certain coins can be issued for a specific purpose to individuals, or capital flow control can be ensured based on the credentials tied to a person's public key, and the PCS signature can attest compliance.

In the context of recent CBDC proposals [50, 64], Platypus [64] is a very elaborate and nuanced system. During payments, where interaction with a central bank is required, it is proposed to carefully distinguish types of transactions and, depending on this, the bank might request further information, in plain or via zero-knowledge proofs. Although being centralized, the system does not admit to prove statements about sender and receiver of a transaction simultaneously, e.g., to control whether cash flow between two individuals is compliant with capital control. In such a scenario, information needs to be revealed to the central bank. However, the approach we take to make this possible in ul-PCS can be directly applied to such centralized designs and enrich them with even more fine-grained policies. The same holds for Peredi [50], which, compared to Platypus, does not put forth a fine-grained transaction model and presents a revocation-based auditability solution.

3 Preliminaries

Notation. We denote the security parameter by λ 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 $\mathsf{negl}(\lambda)$ is called *negligible* if for every positive polynomial $p(\lambda)$, there exists λ_0 such that for all $\lambda > \lambda_0$: $\mathsf{negl}(\lambda) < 1/p(\lambda)$. If clear from the context, we sometimes omit λ for improved readability. The set $\{1, \dots, n\}$ is denoted as $[n]$ for a positive integer n. For the equality check of two elements, we use "=". The assign operator is denoted with ":=", whereas randomized assignment is denoted with $a \leftarrow A$, with a randomized algorithm A and where the randomness is not explicit. If the randomness is explicit, we write $a := A(x; r)$ where x is the input and r is the randomness. For algorithms $\mathcal{A}$ and $\mathcal{B}$, we write $\mathcal{A}^{\mathcal{B}(\cdot)}(x)$ to denote that $\mathcal{A}$ gets x as an input and has black-box oracle access to $\mathcal{B}$, that is, the response for an oracle query q is $\mathcal{B}(q)$.

3.1 Bilinear Group Setup

Some of our schemes require a bilinear group setup. We use multiplicative notation to refer to group operations.

Definition 1 (Bilinear Groups [14]). *An asymmetric bilinear group generator $\mathcal{BG}(1^\lambda)$ returns a tuple $\mathsf{pp} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \mathsf{G}_1, \mathsf{G}_2)$, such that $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of the same prime order p, $\mathsf{G}_1 \in \mathbb{G}_1$ and $\mathsf{G}_2 \in \mathbb{G}_2$ are the generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an efficiently computable bilinear pairing with the following properties;*

– *non-degeneracy:* $e(\mathsf{G}_1, \mathsf{G}_2) \neq 1_{\mathbb{G}_T}$,
– *bilinearity:* $\forall\, a, b \in \mathbb{Z}_p :\ e(\mathsf{G}_1^a, \mathsf{G}_2^b) = e(\mathsf{G}_1, \mathsf{G}_2)^{ab} = e(\mathsf{G}_1^b, \mathsf{G}_2^a)$.

Throughout this work, we rely on Type-III bilinear groups for the distinct cyclic groups $\mathbb{G}_1 \neq \mathbb{G}_2$, where there is no efficient algorithm to compute a nontrivial homomorphism in both directions [39]. This type is known as the most efficient choice.

3.2 Pseudorandom Functions

We recall the definition of a pseudorandom function (PRF) as it has been defined in [41].

Definition 2 (Pseudorandom Function). *A pseudo-random function is a keyed function $\mathsf{PRF} : \{0,1\}^\lambda \times \mathcal{X} \to \mathcal{Y}$, where evaluation is done via an efficient algorithm $\mathsf{PRF.Eval}(\mathsf{k}, x)$. For $\beta \in \{0,1\}$, we define the experiment $\mathrm{IND}_\beta^{\mathsf{PRF}}$ in Figure 2, where the oracle $\mathcal{O}$ is defined as:*

$$\mathcal{O}(x) = \begin{cases} \mathsf{PRF.Eval}(\mathsf{k}, x) & \textit{if } \beta = 0 \\ \mathsf{RF}(x) & \textit{if } \beta = 1 \end{cases} .$$

with $\mathsf{RF}(x)$ denoting a random function. We define the advantage of an adversary $\mathcal{A}$ in the following way:

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

A pseudorandom function PRF is secure, if for any polynomial-time adversary $\mathcal{A}$, there exists a negligible function negl such that: $\mathsf{Adv}_{\mathsf{PRF},\mathcal{A}}^{\mathrm{IND}}(\lambda) \leq \mathsf{negl}(\lambda)$.

$\mathrm{IND}_\beta^{\mathsf{PRF}}(\lambda, \mathcal{A})$
$\mathsf{k} \leftarrow \{0,1\}^\lambda$
$\alpha \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda)$
Output: α

Fig. 2: Security Game for PRF

3.3 Digital Signatures

We recap the definition of digital signatures as well as existential unforgeability [43].

Definition 3 (Digital Signatures). *A digital signature scheme (DS) is a triple of PPT algorithms $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$, defined as follows:*

– $\mathsf{Setup}(1^\lambda)$*: Takes as input a unary representation of the 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 4 (Existential Unforgeability). *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *be a DS scheme. We define the experiment* $\mathsf{EUF\text{-}CMA}^{\mathsf{sig}}$ *in Figure 3 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*

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

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

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

Fig. 3: Existentially Unforgeability for signatures.

3.4 Structure-Preserving Signatures on Equivalence Classes.

Structure-Preserving Signatures (SPS) [1] are a special type of digital signatures defined over bilinear groups that fulfill certain extra properties. More precisely, the verification key, message and signature are only source group elements and, to verify the validity of a signature, only group membership checks and pairing product equations are allowed. SPS have the same algorithm as digital signatures as defined in Definition 3 and guarantee unforgeability as defined in Definition 4.

SPS on Equivalence classes (SPS-EQ) proposed by Hanser and Slamanig [46] are special type of SPS that enable joint re-randomization of signatures and the signed messages. SPS-EQ provide a controlled form of malleability such that one can change the representation of the message and the corresponding signature. More precisely, for a given prime-order group $\mathbb{G}$ we can define a projective vector $(\mathbb{G}^*)^\ell$ based on the following relation, where $\ell > 1$ and $\mathbb{G}^*$ denotes the set of all group elements without the identity element of the group.

$$\mathcal{R} := \left\{ (\vec{M}, \vec{M}^*) \in (\mathbb{G}^*)^\ell \times (\mathbb{G}^*)^\ell \mid \exists\, \mu \in \mathbb{Z}_p^* \text{ s.t. } \vec{M}^* = \vec{M}^\mu \right\} \ . \tag{1}$$

This is an equivalence relation for prime order groups.
The equivalence class of a vector $\vec{M} \in (\mathbb{G}^*)^\ell$ for some $\ell > 1$ is defined by:

$$[\vec{M}]_\mathcal{R} := \{\vec{M}^* \in (\mathbb{G}^*)^\ell \mid (\vec{M}, \vec{M}^*) \in \mathcal{R}\} \ .$$

The Class-hiding property of equivalence classes guarantees that it is computationally hard to distinguish elements of the same equivalence class from randomly sampled group elements.

Definition 5 (Class-Hiding [46]). *A relation $\mathcal{R}$ is called class-hiding if for all PPT adversaries, $\mathcal{A}$, and $\ell > 1$ we have:*

$$\left| \Pr\left[\begin{matrix} \vec{M} \xleftarrow{\$} (\mathbb{G}^*)^\ell, \vec{M}_0 \xleftarrow{\$} (\mathbb{G}^*)^\ell, \vec{M}_1 \xleftarrow{\$} [\vec{M}]_\mathcal{R}, \\ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{A}(\vec{M}, \vec{M}_b) \mid b = b' \end{matrix} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

Hanser and Slamanig [46] formally prove that, as long as DDH is hard, the relation described in equation 1 is class-hiding. We only consider this relation in this work. In our bilinear setting, the message is based on the second group $\mathbb{G}_2$, but we present the scheme in its general form:

Definition 6 (Structure-Preserving Signatures on Equivalence classes [46]). *In an asymmetric bilinear group, a structure preserving signature over (message space) $(\mathbb{G}_i^*)^\ell$ consists of the following PPT algorithms:*

- $\mathsf{Setup}_\mathcal{R}(1^\lambda)$*: The setup algorithm is a probabilistic algorithm that takes the security parameter λ in its unary representation as input. It outputs public parameters* pp *as well as an asymmetric bilinear group.*
- $\mathsf{KeyGen}_\mathcal{R}(\mathsf{pp}, \ell)$*: The key generation algorithm is a probabilistic algorithm that takes the public parameters* pp *and a vector length $\ell > 1$ as inputs. It outputs the key-pair* $(\mathsf{sk}, \mathsf{vk})$.
- $\mathsf{Sign}_\mathcal{R}(\mathsf{pp}, \mathsf{sk}, \vec{M})$*: The signing algorithm is a probabilistic algorithm that takes public parameters* pp*, secret key* sk *and a representative message $\vec{M} \in (\mathbb{G}_i^*)^\ell$ for class $[\vec{M}]_\mathcal{R}$ as inputs. It outputs the signature σ on message $\vec{M}$.*
- $\mathsf{Verify}_\mathcal{R}(\mathsf{pp}, \mathsf{vk}, \vec{M}, \sigma)$*: The verification algorithm is a deterministic algorithm that takes public parameters* pp*, a representative message $\vec{M} \in (\mathbb{G}_i^*)^\ell$, a signature σ and a verification key* vk *as inputs. It then outputs 1 if σ is a valid signature on $\vec{M}$ and 0 otherwise.*
- $\mathsf{ChgRep}_\mathcal{R}(\mathsf{pp}, \vec{M}, \sigma, \mu, \mathsf{vk})$*: The change representation algorithm is a probabilistic algorithm and takes public parameters* pp*, a representative message $\vec{M} \in (\mathbb{G}_i^*)^\ell$, a signature σ, a scalar $\mu \in \mathbb{Z}_p^*$ and the verification key* vk *as inputs. It outputs a randomized signature σ' on a new representative message $\vec{M}' = \vec{M}^\mu$.*

Since in our work all keys are honestly generated we omit the specification of the function that checks whether a private key is consistent with a given public key (since this holds for honestly generated key pairs).

The primary security requirements for a SPS-EQ scheme are *correctness* and *existential unforgeability against chosen message attack*, which are defined as follows:

Definition 7 (Correctness). *A SPS-EQ scheme over* $(\mathbb{G}_i^*)^\ell$ *is called* correct*, if the following holds with overwhelming probability for a valid setup* pp*, any message* $\vec{M} \in (\mathbb{G}_i^*)^\ell$*, any (valid) key pair* $(\mathsf{sk}, \mathsf{pk})$ *in the support of* $\mathsf{KeyGen}_\mathcal{R}(\mathsf{pp}, \ell)$*, and any scalar* $\mu \in \mathbb{Z}_p^*$*:*

$$\Pr\begin{bmatrix}\mathsf{Verify}_\mathcal{R}\left(\mathsf{pp}, \mathsf{vk}, \vec{M}, \mathsf{Sign}_\mathcal{R}(\mathsf{pp}, \mathsf{sk}, \vec{M})\right) = 1 \wedge \\ \mathsf{Verify}_\mathcal{R}(\mathsf{pp}, \mathsf{vk}, \vec{M}^\mu, \mathsf{ChgRep}_\mathcal{R}(\vec{M}, \mathsf{Sign}_\mathcal{R}(\mathsf{pp}, \mathsf{sk}, \vec{M}), \mu, \mathsf{vk})) = 1\end{bmatrix} .$$

Definition 8 (Existential Unforgeability). *A SPS-EQ over* $(\mathbb{G}_i^*)^\ell$ *is called adaptively* $\mathsf{EUF\text{-}CMA}$*-secure if for all PPT adversaries* $\mathcal{A}$ *with access to the signing oracle* $\mathcal{O}_\mathsf{Sign}$ *we have:*

$$\Pr\begin{bmatrix}\mathsf{pp} \leftarrow \mathsf{Setup}_\mathcal{R}(1^\lambda), (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}_\mathcal{R}(\mathsf{pp}, \ell), \left(\vec{M}^*, \sigma^*\right) \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{Sign}}(\mathsf{pp}, \mathsf{vk}) : \\ \forall\ \vec{M} \in \mathcal{Q}^\mathsf{Sign} : [\vec{M}^*]_\mathcal{R} \neq [\vec{M}]_\mathcal{R}\ \wedge\ \mathsf{Verify}_\mathcal{R}\left(\mathsf{pp}, \mathsf{vk}, \vec{M}, \sigma^*\right) = 1\end{bmatrix} \leq \mathsf{negl}(\lambda) ,$$

where the signing oracle $\mathcal{O}_\mathsf{Sign}$ *takes a message* $\vec{M} \in (\mathbb{G}_i^*)^\ell$ *as input, outpus* $\mathsf{Sign}_\mathcal{R}(\mathsf{pp}, \mathsf{sk}, \vec{M})$ *and adds the message to the query set* $\mathcal{Q}^\mathsf{Sign}$*.*

Finally, we require signature adaptation which shows that signature strings can be perfectly randomized (and thus made unlinkable).

Definition 9 (Signature Adaptation). *An SPS-EQ scheme over* $(\mathbb{G}_i^*)^\ell$ *perfectly adapts signatures if for all tuples* $(\mathsf{sk}, \mathsf{pk}, \vec{M}, \sigma, \mu)$*, where* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \ell)$*,* $\vec{M} \in (\mathbb{G}_i^*)^\ell$ *and* $\mathsf{Verify}\left(\mathsf{pp}, \mathsf{vk}, \vec{M}, \sigma\right) = 1$*, the two distributions* $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \vec{M}^\mu)$ *and* $\mathsf{ChgRep}_\mathcal{R}(\mathsf{pp}, \vec{M}, \sigma, \mathsf{vk}, \mu)$ *are identical.*

3.5 A Weak Positive Accumulator

We recall an accumulator construction proposed by Karantaidou and Baldimtsi [48] and consider it in the asymmetric bilinear group setting. The construction is derived from Boneh-Boyen signatures [13] and is based on the q-SDH assumption. We only need to consider the positive accumulator, and thus the accumulator value remains constant. In fact, in our application, we only need to guarantee soundness of the accumulator against a weak adversary. As we show below, the soundness requirement of the accumulator corresponds to what is defined as weakly-unforgeable in [13] for the signature scheme. The witnesses are of constant size, independent of the number of elements in the accumulator set and, additionally, the membership witnesses, after adding new elements, do not need to be updated. In fact, the public accumulator will be set to be the "public key" of the signature scheme and hence does not leak any information about the added elements. The simple accumulator we need can be defined by the following PPT algorithms for the bilinear group setting, where $\mathbb{G}_1 = \langle \mathsf{G}_1 \rangle$, $\mathbb{G}_2 = \langle \mathsf{G}_2 \rangle$. The public parameters are $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, \mathsf{G}_1, \mathsf{G}_2, e)$.

– $\mathsf{ACC.Create}(\mathsf{pp})$: Sample $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$ and define $\mathsf{A} \leftarrow \mathsf{G}_2^\alpha$ and $\mathsf{msk} \leftarrow \alpha$ and return $(\mathsf{A}, \mathsf{msk})$. The accumulator domain is $\mathcal{D} = \mathbb{Z}_p \setminus \{\alpha\}$.

– $\mathsf{ACC.Add}(\mathsf{pp}, \mathsf{A}, \mathsf{msk}, x)$: To add a new element $x \in \mathcal{D}$ to the accumulator, parse $\mathsf{msk} = \alpha \in \mathbb{Z}_p^*$ and check that $\mathsf{A} = \mathsf{G}_2^\alpha$. If the check succeeds compute and return the witness $w_x = \mathsf{G}_1^{1/(x+\alpha)}$.
– $\mathsf{ACC.MemVrf}(\mathsf{pp}, \mathsf{A}, x, w_x)$: If the equation $e(\mathsf{G}_1, \mathsf{G}_2) = e(w_x, \mathsf{A}\mathsf{G}_2^x)$ holds, return 1 to approve the membership of x in the accumulator with value A; otherwise output 0 to reject it.

It is straightforward to see that the accumulator is correct. For our purposes, the accumulator has to satisfy the weak soundness notion w.r.t. public parameters pp as we define it in Figure 4. Note that this is a definition tailored to our problem which simplifies the proof of the overall scheme.

$\mathsf{W\text{-}SND}(\mathsf{pp}, \mathsf{ACC}, \mathcal{A})$
$(x_1, \ldots, x_q, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{A}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\mathsf{pp})$
Compute for all $i \in [q] : \pi_i \leftarrow \mathsf{ACC.Add}(\mathsf{pp}, \mathsf{A}, \mathsf{msk}, x_i)$
$(x^*, \pi^*) \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{A}, (\pi_1, \ldots, \pi_q))$
$\mathsf{return} \Big(\forall i : x^* \neq x_i\Big) \wedge \Big(e(\mathsf{G}_1, \mathsf{G}_2) = e(\pi^*, \mathsf{A}\mathsf{G}_2^x)\Big)$

Fig. 4: A weak soundness notion for the accumulator.

We state the following lemma relating the concrete security of q-SDH to the (concrete) winning probability of the above game.

Lemma 1. *Let* $\mathsf{pp} = (p, \mathsf{G}_1, \mathsf{G}_2, e)$ *be the public parameters. For any PPT adversary* $\mathcal{A}$, *asking at most* q *queries, that wins the game* $\mathsf{W\text{-}SND}(\mathsf{pp}, \mathsf{ACC}, \mathcal{A})$ *with probability* ε, *there is a PPT adversary* $\mathcal{A}'$ *that on input the* q*-SDH instance* $(\mathsf{G}_1, y\mathsf{G}_1, \ldots, y^{q'}\mathsf{G}_1, \mathsf{G}_2, y\mathsf{G}_2)$, *where* $y \in \mathbb{Z}_p^*$ *is sampled uniformly at random, returns a valid solution* $(c, (y+c)^{-1}\mathsf{G}_1)$ *for some* $c \in \mathbb{Z}_p \setminus \{-y\}$, *with probability* ε *as long as* $q \leq q'$ *(where the probability is taken over the random choice of* y *and the internal randomness of* $\mathcal{A}'$*).*

Proof. The proof follows directly from the security proof of the weakly-secure short signature scheme in [13] (version 2014) by observing that ACC is just the weakly-secure signature scheme in disguise, and that our soundness notion perfectly matches the notion of weak unforgeability of [13]. $\square$

Note that, since the statement holds for any concrete set of parameters, it also holds over any distribution of parameters and thus we obtain the asymptotic statement that the accumulator is sound, except with negligible probability in λ under the q-SDH assumption, relative to the bilinear group generation algorithm $\mathcal{BG}(1^\lambda)$ generating the parameters pp.

3.6 Public-Key Encryption

Now, we introduce public-key encryption, together with the notion of IND-CPA security.

Definition 10 (Public-Key Encryption). *A* public-key encryption *(PKE) scheme is a tuple of three algorithms* $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Setup}(1^\lambda)$*: Takes as input a unary representation of the security parameter* λ *and outputs a public key* pk *and a secret key* sk*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*: Takes as input the public key* pk *and a message* $m \in \mathcal{M}$*, and outputs a ciphertext* ct*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: Takes as input the secret key* sk *and a ciphertext* ct *and outputs a message* m' *or* $\perp$*.*

A public-key encryption scheme PKE *is* correct *if for all* $\lambda \in \mathbb{N}$*, and for all key-pairs* $(\mathsf{pk}, \mathsf{sk})$ *in the support of* $\mathsf{Setup}(1^\lambda)$*, we have*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] = 1.$$

In this work, we give the adversary access to an encryption challenge oracle that can be queried using multiple challenge message pairs (m_0, m_1). This security definition follows from the standard security definition for a single challenge query using a simple hybrid argument.

Definition 11 (Indistinguishability-Based Chosen-Plaintext Security). *Let* $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme as defined above. For* $\beta \in \{0, 1\}$*, we define the experiment* $\text{IND-CPA}_\beta^{\mathsf{PKE}}$ *in Figure 5, where the left-or-right oracle is defined as:*

$\mathsf{QEncLR}_\beta(\cdot, \cdot)$**:** *On input two messages* m_0 *and* m_1*, output* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, m_\beta)$*.*

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

$$\mathsf{Adv}_{\mathsf{PKE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) = |\Pr[\text{IND-CPA}_0^{\mathsf{PKE}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\text{IND-CPA}_1^{\mathsf{PKE}}(1^\lambda, \mathcal{A}) = 1]|.$$

A predicate-only predicate encryption scheme PKE *is called* IND-CPA secure *if for any valid polynomial-time adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that* $\mathsf{Adv}_{\mathsf{PKE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \mathsf{negl}(\lambda)$*.*

$\text{IND-CPA}_\beta^{\mathsf{PKE}}(1^\lambda, \mathcal{A})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEncLR}_\beta(\cdot,\cdot)}(\mathsf{pk})$
Output: α

Fig. 5: IND-CPA security game of PKE.

3.7 Predicate Encryption

To allow for oblivious policy evaluations, we also recap the notion of *predicate-only predicate encryption* as it has been introduced in Katz et al. [49].

Definition 12 (Predicate-Only Predicate Encryption). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of sets $\mathcal{F}_\lambda$ of predicates $f\colon \mathcal{X}_\lambda \to \{0,1\}$. A* predicate-only predicate encryption *(PE) scheme for the functionality class $\mathcal{F}_\lambda$ is a tuple of four algorithms $\mathsf{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$:*

- $\mathsf{Setup}(1^\lambda)$*: Takes as input a unary representation of the security parameter λ and outputs the master public key mpk and the master secret key msk.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: Takes as input the master secret key msk and a function $f \in \mathcal{F}$, and outputs a functional key sk_f.*
- $\mathsf{Enc}(\mathsf{mpk}, x)$*: 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*

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

In the initial work of Katz et al. [49], the authors only introduce the notion of selective security. The corresponding indistinguishability based adaptive security notion for predicate encryption has been introduced in [55]. We present a modification of this definition where the adversary has access to a challenge oracle to which it can submit multiple challenges instead of being able to only submit a single challenge. This security definition directly follows from the standard security definition using a simple hybrid argument.

Definition 13 (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 $\mathsf{AH}^{\mathsf{PE}}_\beta$ in Figure 6, where the left-or-right oracle is defined as:*

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

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

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

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

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

Fig. 6: Attribute-Hiding game of PE.

3.8 Non-interactive Zero-Knowledge Proofs

In this section, we introduce the notion of non-interactive zero knowledge (NIZK) proofs [10, 36, 42].

Definition 14 (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 the unary representation of the security parameter λ and outputs a 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:

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

Fig. 7: Zero-knowledge property of NIZK.

Definition 15 (Zero-Knowledge). *Let* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a NIZK proof system for a relation R and the corresponding language L, $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ a pair of algorithms* *(the simulator), with* $\mathcal{S}'(\mathsf{CRS}, \tau, x, w) = \mathcal{S}_2(\mathsf{CRS}, \tau, x)$ *for* $(x, w) \in R$*, and* $\mathcal{S}'(\mathsf{CRS}, \tau, x, w) =$ $\mathsf{failure}$ *for* $(x, w) \notin R$*. For* $\beta \in \{0, 1\}$*, we define the experiment* $\mathsf{ZK}_\beta^{\mathsf{NIZK}}(1^\lambda, \mathcal{A})$ *in Figure 7. The associated advantage of an adversary* $\mathcal{A}$ *is defined as*

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

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

Besides zero-knowledge and soundness, we rely on the notion of extractability [22].

Definition 16 (Extractability). *Let* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a NIZK proof system for a relation* R *and the corresponding language* L*, let* $\mathsf{Ext} = (\mathsf{Ext}_1, \mathsf{Ext}_2)$ *be a pair of algorithms (the extractor). We define the extraction advantages of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}_{\mathsf{NIZK},\mathcal{A}}^{\mathrm{CRS}} \coloneqq |\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda); 1 \leftarrow \mathcal{A}(\mathsf{CRS})] - \Pr[(\mathsf{CRS}, \mathsf{st}) \leftarrow \mathsf{Ext}_1(1^\lambda); 1 \leftarrow \mathcal{A}(\mathsf{CRS})]|,$$

and

$$\mathsf{Adv}_{\mathsf{NIZK},\mathcal{A}}^{\mathsf{Extract}}(\lambda) \coloneqq \Pr\left[\begin{matrix}(\mathsf{CRS}_{\mathsf{Ext}}, \mathsf{st}_{\mathsf{Ext}}) \leftarrow \mathsf{Ext}_1(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{CRS}_{\mathsf{Ext}})\end{matrix}; \begin{matrix}\mathsf{Verify}(\mathsf{CRS}_{\mathsf{Ext}}, x, \pi) = 1 \wedge \\ R(x, \mathsf{Ext}_2(\mathsf{CRS}_{\mathsf{Ext}}, \mathsf{st}_{\mathsf{Ext}}, x, \pi)) = 0\end{matrix}\right]$$

A NIZK proof system NIZK *is called extractable, with respect to an extractor* $\mathsf{Ext} =$ $(\mathsf{Ext}_1, \mathsf{Ext}_2)$*, if* $\mathsf{Adv}_{\mathsf{NIZK},\mathcal{A}}^{\mathrm{CRS}} \leq \mathsf{negl}(\lambda)$ *and* $\mathsf{Adv}_{\mathsf{NIZK},\mathcal{A}}^{\mathsf{Extract}}(\lambda) \leq \mathsf{negl}(\lambda)$*. Additionally, we call an extractable non-interactive zero-knowledge proof a non-interactive zero-knowledge proof of knowledge (NIZKPoK).*

4 Unlinkable PCS: Model and Security Requirements

Now, we present the syntax of unlinkable policy-compliant signatures (ul-PCS). ul-PCS are basically defined as PCS [6] with the only difference that they contain an additional rerandomization algorithm that allows for the rerandomization of key pairs.

Definition 17 (Unlinkable Policy-Compliant Signatures). *Let* $\{X_\lambda\}_{\lambda \in \mathbb{N}}$ *be a family of attribute sets and denote by* $\mathcal{X}_\lambda$ *the powerset of* X_λ*. Further let* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ *be a family of sets* $\mathcal{F}_\lambda$ *of predicates* $F \colon \mathcal{X}_\lambda \times \mathcal{X}_\lambda \to \{0, 1\}$*. Then an* unlinkable policy-compliant signature (ul-PCS) *scheme for the functionality class* $\mathcal{F}_\lambda$ *is a tuple of four PPT algorithms* $\mathsf{ULPCS} =$ $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RandKey}, \mathsf{Sign}, \mathsf{Verify})$*:*

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

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

$\mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk})$**:** *On input the master public key* mpk *and a secret key* sk*, output a new public-secret-key pair* $(\mathsf{pk}', \mathsf{sk}')$*.*

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

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

Remark 1 (On distributing Setup *and* KeyGen *for the DAP use case).* The described algorithms correspond to the intended use-case sketched in Figure 1. The signature scheme, in fact, can be used to sign any message towards some recipient's public key. Note that the registration process of a party with attributes x is abstracted as a simple key-generation procedure to be able to focus on the core technical challenges of the construction and does not indicate that a single entity necessarily stands behind that. In fact, in a typical application, we do not need the credential authority in the operational phase (after user enrollment) and it does not need to know users' keys, which makes Setup and KeyGen suitable for distributed implementations without impacting the operational efficiency. In this setting, msk would be shared among n servers at the price of a more expensive enrollment. All our concrete schemes in later sections admit distributed implementations of Setup and KeyGen based on standard techniques (cf. Section 7.3.). This is important because if, for example, the whole master secret key is leaked there is no hope to achieve a reasonable notion of privacy for the attributes: the adversary can, with knowledge of the master secret key, generate keys by itself and consequently use them to check w.r.t. any public key in the system whether or not valid signatures can be generated. This shows that credential issuance must not be corrupted in order to achieve a reasonable notion of privacy.

Remark 2 (On public vs. private re-randomization). The interface of RandKey is such that every party is in charge of its own re-randomizations. Alternatively, one could imagine a public re-randomization, that allows to randomize any given valid public key, yielding a new valid public key. We observe that the immediate benefit of such a public re-randomization does not appear to be substantial—given that the additional constraints have serious implications on the practical feasibility of PCS (as we explain further below).

Note that in the anticipated use-cases in payment systems, it is typical that cautious users maintain new keys across payment sessions or go even as far as using an address only once by default or for increased security (cf. Section 7). In addition, the sender in a payment session must determine the receiving address (i.e., public key) by some "off-chain" mechanism, at which point a receiver can present its (re-randomized) key. The benefit of public re-randomization appears to be in saving some bandwidth if the sender casts multiple transactions in one session, compared to the straightforward solution where a receiver—having precomputed in an offline phase a bunch of ranodomized keys—presents multiple receiving addresses to the sender at the beginning of a payment session. Furthermore, across payment sessions, a similar improvement is only achieved, if at all, if the recipient is willing to link its payment session to the current one in the offchain communication phase, and only if this is more efficient than just presenting a new key, in which case one can simply present a randomized version.

For this seemingly slight improvement, one would pay a rather hight price in terms of practical feasibility for PCS: it follows from the relationship between predicate encryption

$\underline{\mathsf{CORR}^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A})}$	$\underline{\mathcal{O}_{\mathsf{Gen}}(x)\text{:}}$
$(F, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$	$c \leftarrow c + 1$
$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda, F);\quad c \leftarrow 0$	$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x)$
$((i,j),(k,\ell),m) \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Gen}}(\cdot),\mathcal{O}_{\mathsf{ReRand}}(\cdot)}(\mathsf{st}, \mathsf{mpk})$	$Q_c \leftarrow [(\mathsf{sk}, \mathsf{pk}, x)]$
$(\mathsf{sk}_S, \mathsf{pk}_S, x_S) \leftarrow Q_i[j]$	Return $(\mathsf{sk}, \mathsf{pk})$
$(\mathsf{sk}_R, \mathsf{pk}_R, x_R) \leftarrow Q_k[\ell]$	$\underline{\mathcal{O}_{\mathsf{ReRand}}(j)\text{:}}$
$b \leftarrow \mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S, \mathsf{pk}_R, m,$	If $j > c$ return $\perp$
$\quad\quad \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}_S, \mathsf{pk}_R, m))$	$(\mathsf{sk}, \mathsf{pk}, x) \leftarrow Q_j[\lvert Q_j\rvert]$
Return $b \neq F(x_S, x_R)$	$(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk})$
	$Q_j \leftarrow Q_j \vert\vert (\mathsf{sk}', \mathsf{pk}', x)$
	Return $(\mathsf{sk}', \mathsf{pk}')$

Fig. 8: Correctness Experiment of a ul-PCS scheme.

and PCS (cf. Section 2) that such a public re-randomization feature is much more difficult to achieve, as it does not only imply re-randomizable predicate-encryption. Furthermore, we also require that such a PE scheme comes with enough structural properties to enable transferability of the validity assurance (such as achieved by SPS-EQ). It is an interesting open question whether such PE schemes can be constructed efficiently based on standard assumptions. The same reasoning applies to the more specialized policies, where the above considerations put additional constraints on practical realizability. In contrast, we are able to construct and implement PCS as defined above using standard tools and are able to give a concrete prototype for all proposed schemes.

4.1 Correctness and Detectability Relation

Correctness. A ul-PCS scheme is correct if in any execution, honestly generated signatures computed using honestly generated private and public keys, potentially re-randomized multiple times, reflect the policy. Compared to standard PCS, it is easier to capture this as a correctness experiment, since the interaction introduced with re-randomization leads to more complex scenarios. A ul-PCS scheme is called correct if for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in experiment CORR, specified in Figure 8, the probability $\Pr[\mathsf{CORR}^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A}) = 1]$ is negligible in the security parameter.

Detectability Relation. Compared to the requirements of a standard PCS scheme, the requirements for an unlinkable PCS scheme pose a definitional challenge: we need to capture unforgeability and policy-compliance in a security game but, at the same time, keys are randomized (potentially by the adversary) and no efficient algorithm could detect whether this is in fact a valid forgery—since all attributes are private and parties are not traceable. We solve this definitional issue by introducing what appears to be a quite natural requirement:

$\mathsf{Det}^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A})$	$\mathcal{O}_{\mathsf{Gen}}(x)$:
$(F, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$	$c \leftarrow c + 1$
$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda, F)$	$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x)$
$c := 0$	$Q_c \leftarrow [(\mathsf{sk}, \mathsf{pk})]$
$(i, j) \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Gen}}, \mathcal{O}_{\mathsf{ReRand}}}(\mathsf{st}, \mathsf{mpk})$	Return $(\mathsf{sk}, \mathsf{pk})$
$(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow Q_i[j]$	$\mathcal{O}_{\mathsf{ReRand}}(j)$:
$i^* \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}^*, (Q_1, \ldots, Q_c))$	If $j > c$, return $\perp$
Return $i^* \neq i$	$(\mathsf{sk}, \mathsf{pk}) \leftarrow Q_j[\lvert Q_j \rvert]$
	$(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk})$
	$Q_j \leftarrow Q_j \| (\mathsf{sk}', \mathsf{pk}')$
	Return $(\mathsf{sk}', \mathsf{pk}')$

Fig. 9: Detectability Experiment of a ul-PCS scheme.

any ul-PCS scheme must satisfy a detectability relation which intuitively captures the property that a party, knowing its own initial secret key, can detect whether a valid public key is in fact derived from it (that is, a party can detect its own public keys in a ledger). Using this detection property, the challenger in the security game can determine which keys belong to which oracle queries. The algorithm is called Detect, and takes as input a target public key, and the keys generated by the challenger for different parties. The algorithm must return the index of the party that the target key belongs to. Looking ahead, this must hold even if the adversary is in charge of re-randomizations. Clearly, such an algorithm must satisfy a non-triviality condition: when keys are honestly generated and re-randomized, the algorithm detects only correct relations and never confuses parties.[6]

Let Detect be an algorithm that takes as input the master public key mpk, a candidate key pk^*, and a list consisting of sequences of key pairs $(Q_1, \ldots, Q_c)$, and outputs an index or $\perp$. A ULPCS scheme is said to have the detectability property if there is an efficiently computable algorithm Detect such that for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in experiment DTCT, specified in Figure 9, the probability $\Pr[\mathsf{DTCT}^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A}) = 1]$ is negligible in the security parameter.

We point out that this form of detectability is very different from tracing—the property or feature that an additional entity, the auditor, is able to trace parties by means of a special viewing or revocation key. This property is not entirely new and has already been introduced in the context of, e.g., Monero [3]. In Section 7, we give more details on how to embed ul-PCS in larger contexts and discuss the traceability requirement appearing in the literature on CBDCs.

[6] It is instructive to observe that such (private-key) detectability relations are also studied in the context of RCCA variants [7].

4.2 Adversarial Capabilities in the Security Games

Before presenting the notions of unforgeability, attribute-hiding and unlinkability, we describe the adversarial capabilities in the different security games.

To keep track of all honestly generated keys, corrupted keys and generated signatures, we define the initially empty sets $\mathcal{QK}$, $\mathcal{QC}$ and $\mathcal{QS}$, respectively.

Key-Generation Oracle $\mathsf{QKeyGen}(\cdot)$**:** On the i-th input of an attribute set x_i, generate $(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x_i)$, and add $((i, 0), \mathsf{pk}_i^0, \mathsf{sk}_i^0, x_i)$ to $\mathcal{QK}$. Return pk_i^0.

Left-or-Right Key-Generation Oracle $\mathsf{QKeyGenLR}_\beta(\cdot, \cdot)$**:** On the i-th input of a pair of attribute sets $x_{i,0}$ and $x_{i,1}$, generate $(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x_{i,\beta})$, add $((i, 0), \mathsf{pk}_i^0, \mathsf{sk}_i^0, x_{i,0}, x_{i,1})$ to $\mathcal{QK}$, and return pk_i^0.

Key-Randomization Oracle $\mathsf{QRandKey}(\cdot)$**:** On input an index i, if $\mathcal{QK}$ contains entries $((i, 0), \mathsf{pk}_i^0, \mathsf{sk}_i^0, \dots), \dots, ((i, c_i), \mathsf{pk}_i^{c_i}, \mathsf{sk}_i^{c_i}, \dots)$, then compute $(\mathsf{pk}_i^{c_i+1}, \mathsf{sk}_i^{c_i+1}) \leftarrow \mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk}_i^{c_i})$ and add $((i, c_i + 1), \mathsf{pk}_i^{c_i+1}, \mathsf{sk}_i^{c_i+1}, \dots)$ to $\mathcal{QK}$ and return $\mathsf{pk}_i^{c_i+1}$.

Corruption Oracle $\mathsf{QCor}(\cdot)$**:** On input an index i, if $\mathcal{QK}$ contains entries $((i, j), \mathsf{pk}_i^j, \mathsf{sk}_i^j, \dots)$ for $0 \leq j \leq c_i$ for some $c_i \geq 0$, then copy these entries from $\mathcal{QK}$ to $\mathcal{QC}$ and return the list $(\mathsf{sk}_i^0, \dots, \mathsf{sk}_i^{c_i})$.

Signing Oracle $\mathsf{QSign}(\cdot, \cdot, \cdot)$**:** On input an index pair (i, j), a public key pk' and a message m, if $\mathcal{QK}$ contains an entry $((i, j), \mathsf{pk}_i^j, \mathsf{sk}_i^j, \dots)$, then compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}_i^j, \mathsf{pk}', m)$, add $((i, j), \mathsf{pk}_i^j, \mathsf{pk}', m, \sigma)$ to $\mathcal{QS}$ and return the signature.

Randomization-Challenge Oracle $\mathsf{QRandKey}_\beta(\cdot)$**:** On receiving a query i, do the following: if $\beta = 0$ then set $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk})$, and if $\beta = 1$ set $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x)$, where x is taken from the entry $((i, 0), \mathsf{pk}, \mathsf{sk}, x)$ of $\mathcal{QK}$, and sk is taken from the entry $((i, j), \mathsf{pk}, \mathsf{sk}, x)$ of $\mathcal{QK}$ with highest j for the given i. Add $((i, j + 1), \mathsf{pk}', \mathsf{sk}')$ to $\mathcal{QK}$ and return pk'.

Notice that the randomization-challenge oracle is one way of formalizing key evolution. Follow our application story, we assume a party updates its most recent key (similar to key-evolving signatures). Other equivalent options are possible as well. For notational convenience, if the set $\mathcal{QK}$ contains the sequence $((i, 0), \mathsf{pk}_i^0, \mathsf{sk}_i^0, \dots), \dots, ((i, c_i), \mathsf{pk}_i^{c_i}, \mathsf{sk}_i^{c_i}, \dots)$ we denote by $\mathcal{QK}_i$ the sequence of keys $[(\mathsf{pk}_i^0, \mathsf{sk}_i^0), \dots, (\mathsf{pk}_i^{c_i}, \mathsf{sk}_i^{c_i})]$ of party i.

4.3 Security of ul-PCS

Unforgeability. Unforgeability captures the property that signatures by honest parties cannot be forged and that it is not possible to create valid signatures that are not policy-compliant. In more detail, an adversary $\mathcal{A}$ creates a valid forgery if: (a) it is able to generate a valid signature for a public key belonging to an honest/uncorrupted party, or (b) it is able to generate a valid signature for a key that has never been issued for an attribute set, or (c) it is able to generate a valid signature for a key pair $\mathsf{pk}_S, \mathsf{pk}_R$ where the corresponding attributes do not fulfill the policy F. We capture all these conditions in the security game in Figure 10, which is based on the unforgeability game of [6], incorporating the modifications mentioned above. To efficiently evaluate condition (c), we make use of the mentioned

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

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

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

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

Let $i_{\max}$ be the number of queries made to $\mathsf{QKeyGen}(\cdot)$

$S \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}, (\mathcal{QK}_1, \ldots, \mathcal{QK}_{i_{\max}}))$

$R \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}^*, (\mathcal{QK}_1, \ldots, \mathcal{QK}_{i_{\max}}))$

Let x_S and x_R denote the attributes in case $S, R \neq \perp$

Output: $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge$
$\Big[\big[\exists (i,j), \mathsf{sk}, x \; \forall (i',j'), \sigma : ((i,j), \mathsf{pk}, \mathsf{sk}, x) \in \mathcal{QK} \setminus \mathcal{QC} \wedge$
$((i',j'), \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma) \notin \mathcal{QS}\big] \vee \big[(S \neq \perp) \wedge (R \neq \perp) \Rightarrow F(x_S, x_R) = 0\big]\Big]$

Fig. 10: Unforgeability Game of ULPCS.

detection algorithm. In more detail, to check if the attributes associated with pk and pk^* of a potential forgery output $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ do not fulfill the policy, the attributes associated with pk and pk^* first need to be determined. This is not necessarily straightforward since the keys pk and pk^* might not be generated by the authority but by the adversary through rerandomization. The detection algorithm is used by the challenger to map these keys back to the key-generation event to determine their associated attribute sets.

Definition 18 (Existential Unforgeability of a PCS Scheme). *Let* ULPCS = (Setup, KeyGen, Sign, Verify) *be a ul-PCS scheme that satisfies the detectability property. We define the experiment* $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}$ *in Figure 10, where all oracles are defined as in Section 4.2. The advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is defined by*

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

Such a ul-PCS scheme ULPCS *is called* existential unforgeable under adaptive chosen message attacks *or* existential unforgeable *for short if for any polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{ULPCS},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$*. We further call a ul-PCS scheme* T_{Rand}*-unforgeable if the number of key rerandomization queries* q *is less than* T_{Rand}*, i.e.* $q < T_{\mathsf{Rand}}$*.*

Attribute-Hiding. Attribute hiding captures the strong property that the system does not leak anything about honest parties' attributes except, of course, what can be inferred from legitimate signatures that, published in the system. To capture this, we define a security experiment based on [6], where we, again, have to include an additional oracle and, to make the definition well-defined, need to rely on the detect relation.

The adversary has access to different oracles: (1) a challenge oracle, to which it can submit an attribute pair (x_0, x_1) and receives as a reply the public key pk corresponding to x_β, for β chosen by the challenger; (2) a rerandomization oracle, to which it can submit indices i and then receives as a reply the rerandomization of the public key that corresponds to this index; (3) a corruption oracle, to which it can submit an index and then receives as a reply the secret key that corresponds to the public key associated with the index; and (4) a signing oracle, to which the adversary can submit an index pair (i, j) as well as a public key pk and a message m and then receives as a reply a signature generated using the j'th rerandomization of the i'th secret key for the public key pk over the message m. The goal of the adversary in this game is to determine the bit β, and thus to observe a difference between the two settings. If the success probability of any adversary for both cases, i.e. $\beta = 0$ and $\beta = 1$, is approximately the same then we say that the scheme is secure.

To prevent the adversary from trivially winning the game, we need to specify validity requirements that exclude those distinguishing strategies that are simply based on how the system is supposed to operate (i.e., correctness) [57] (see also Remark 1). First, the adversary is only allowed to ask for the corruption of an index i, if the challenge query for this index consists of the same attribute sets, i.e. $x_0 = x_1$. Second, the adversary is only allowed to ask signing queries for an index (i, j) and receiver key pk such that it holds that $F(x_0, y_0) = F(x_1, y_1)$ where (x_0, x_1) is the i'th key challenge query and (y_0, y_1) are the possible attributes associated with pk. To determine the attributes (y_0, y_1) of pk, we make, again, use of the detectability of the scheme and execute the detection algorithm using pk as an input. The game is formally described in Figure 11.

Definition 19 (IND-Based Attribute Hiding). *Let* $\mathsf{ULPCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a ul-PCS scheme that satisfies the detectability property. For* $\beta \in \{0, 1\}$*, we define the experiment* $\mathsf{AH}_\beta^{\mathsf{ULPCS}}$ *in Figure 11, where all oracles are defined as in Section 4.2. The advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is defined by*

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

We call an adversary valid *if all of the following hold with probability* 1 *over the randomness of the adversary and all involved algorithms, where* $i_{\max}$ *denotes an upper bound on the number of queries to* $\mathsf{QKeyGenLR}_\beta$*:*

- *for every* $((i, j), \mathsf{pk}_i^j, \mathsf{sk}_i^j, x_{i,0}, x_{i,1}) \in \mathcal{QC}$ *and for all* $((k, \ell), \mathsf{pk}_k^\ell, \mathsf{sk}_k^\ell, x_{k,0}, x_{k,1}) \in \mathcal{QK}$ *we have* $x_{i,0} = x_{i,1} =: x_i$ *and* $F(x_i, x_{k,0}) = F(x_i, x_{k,1})$*,*
- *and for all* $((i, j), \mathsf{pk}_i^j, \mathsf{pk}, m, \sigma) \in \mathcal{QS}$*,* $R \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}, (\mathcal{QK}_1, \ldots, \mathcal{QK}_{i_{\max}}))$*, and* $((i, j), \mathsf{pk}_i, \mathsf{sk}_i, x_{i,0}, x_{i,1}) \in \mathcal{QK}$*, we either have* $R = \perp$ *or otherwise* $F(x_{i,0}, x_{R,0}) = F(x_{i,1}, x_{R,1})$ *holds.*

Such a ul-PCS scheme ULPCS *is called* attribute hiding *if for any valid polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\mathsf{AH}}_{\mathsf{ULPCS},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$*. We call a ul-PCS scheme* T_{Rand}*-attribute-hiding if the number of key rerandomization queries* q *is less than* T_{Rand}*, i.e.* $q < T_{\mathsf{Rand}}$*. Finally, we call a ul-PCS scheme outsider-attribute-hiding (outsider-AH) if the adversary does not have access to the corruption oracle.*

Outsider security models an attacker who is just analyzing a transaction graph [21, 24].

$$\boxed{\begin{array}{l}\underline{\mathsf{AH}_\beta^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A})} \\ (F, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{ULPCS.Setup}(1^\lambda, F) \\ \alpha \leftarrow \mathcal{A}_2^{\mathsf{QKeyGenLR}_\beta(\cdot,\cdot),\mathsf{QRandKey}(\cdot),\mathsf{QCor}(\cdot),\mathsf{QSign}(\cdot,\cdot,\cdot)}(\mathsf{st}, \mathsf{mpk}) \\ \textbf{Output: } \alpha\end{array}}$$

Fig. 11: The Attribute-Hiding game for ULPCS.

Unlinkability. Unlinkability captures the property that a party can re-randomize its key such that it is not possible to tell afterwards whether this party is acting again or whether it is another party that freshly joined the system. Coupled with attribute-hiding, this leads to strong privacy guarantees: observing a signature between two freshly re-randomized public keys does not reveal anything beyond the assertion that the attributes behind the keys satisfy the policy without any link to a party's other actions in the system.

The simple single user unlinkability security game (for the case $|\mathcal{QK}| = 1$) in Figure 12 is parameterized by the challenge bit β. It formalizes that an adversary is not able tell apart a user that evolves its key from fresh keys with the same attribute, while all "versions" of the original key are in use to sign off arbitrary messages towards arbitrary recipients (the adversary can create arbitrary users and know their secrets). We show in Appendix A that this intuitive game is sufficient to imply security for the multi-user setting.

Definition 20. *Let* ULPCS $=$ (Setup, KeyGen, Sign, Verify) *be a ul-PCS scheme that satisfies the detectability property. For $\beta \in \{0,1\}$, we define the experiment* $\mathsf{Link}_\beta^{\mathsf{ULPCS}}$ *in Figure 12, where all oracles are defined as in Section 4.2. The advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined by*

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

We call such an ul-PCS scheme ULPCS unlinkable *if for any polynomial-time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\mathsf{Link}}_{\mathsf{ULPCS},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

We call a ul-PCS scheme T_{Rand}-unlinkable if the number of key rerandomization queries q is less than T_{Rand}, i.e. $q < T_{\mathsf{Rand}}$.

On first sight, one might think that this is not the strongest notion for unlinkability since it only captures the unlinkability of a single user. In Appendix A, we show that an unlinkability notion for multiple users is directly implied by the presented single key unlinkability notion.

$$
\begin{array}{|l|}
\hline
\underline{\mathsf{Link}_{\beta}^{\mathsf{ULPCS}}(1^{\lambda}, \mathcal{A})} \\
(F, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda) \\
(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^{\lambda}, F) \\
(x, \mathsf{st}_2) \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(\mathsf{mpk}, \mathsf{st}_1) \\
(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x);\ \mathcal{QK} \leftarrow ((1,0), \mathsf{pk}, \mathsf{sk}, x) \\
\alpha \leftarrow \mathcal{A}_3^{\mathsf{KeyGen}(\mathsf{msk}, \cdot), \mathsf{QRandKey}_{\beta}(\cdot), \mathsf{QSign}(\cdot,\cdot,\cdot)}(\mathsf{pk}, \mathsf{st}_2) \\
\textbf{Output: } \alpha \\
\hline
\end{array}
$$

Fig. 12: Single-Challenge Unlinkability game of ULPCS.

Looking ahead, our concrete schemes achieve T_{Rand} bounded security and in the instantiations we set $T_{\mathsf{Rand}} = 2^{16} - 1$.

5 Unlinkable Policy-Compliant Signature Schemes

In this section, we present our unlinkable policy-compliant signature schemes. We start by presenting the scheme for general policies.

5.1 ul-PCS for Generic Policies

The main idea of the generic scheme in Figure 14 is best motivated by looking at the structure of public and private keys. We equip the public key with an encryption of the user's attributes that can be re-encrypted by the user. For this we use a predicate-encryption scheme that supports the predicate class $f_x(y) := F(x, y)$, where F is the policy (intuitively, a party must be able to evaluate whether it can send to someone else, which is what f_x represents). There must be, however, a link to the issuance of attribute x towards a user by the certification authority (CA). This link is established via pseudorandom identifiers that are developed over sequences of different public-key versions based on a PRF key signed by the CA. This PRF key is part of the private key (a similar technique has been presented in [28] but targets a slightly different goal). It is not only bound to the attributes, but also to the functional key of the PE scheme, and to a signature public key—a master public key that grants the user the right to re-issue fresh signature public keys, akin to an identity-based signature scheme, for itself. Those fresh keys can be published as part of the public key, and signed with the master signature private key Figure 13. Note that due to the unlinkability requirement, the master signature keys as well as the generated signatures must remain private. The NIZK proof in the public key assures the well-formedness of the key. Therefore, the described public key becomes re-randomizable in an unlinkable way. Since we are using a standard PRF, we must further limit the evaluation range (which is the parameter T_{Rand} of the scheme) since, otherwise, we cannot bound the collision probability sufficiently (which is needed to ensure a safe link between attributes and keys).

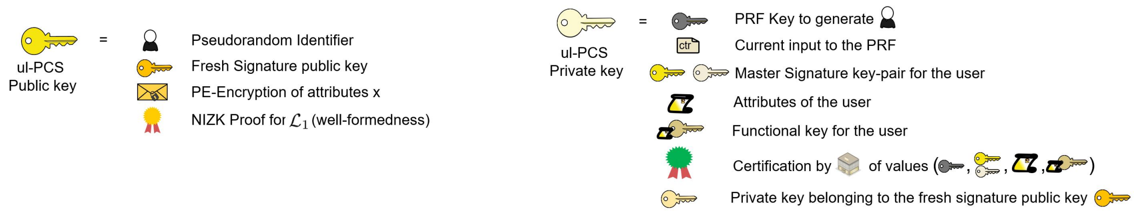

Fig. 13: Key formats of the ul-PCS scheme for the generic construction.

For signing and verification, things are a bit simpler. In order to sign a message towards a recipient, the sender first checks that the recipient's public key is valid and, second, proves that it is eligible to sign towards that recipient by proving that that it has a functional key sk_{f_x} of the PE scheme (signed by the CA) that is able to decrypt the ciphertext in the recipient's public key. By definition of the PE scheme, the ability to decrypt with such a key implies $f_x(y) \coloneqq F(x, y) = 1$ which matches the signing requirement. Finally, the user signs the message and the proof using the fresh signing key. Note that the proofs require public parameters that we refer to as mpk in the formal description, which, for instance, contain the CRSs for the NIZKs and the public key of the PE scheme. We require two languages for the NIZK systems, each for a specific purpose:

- $\mathcal{L}_1$: Used to prove the correct formation of the public key, based on the attributes issued by the authority. We depict this language formally below, which is used during the re-randomization of keys.
- $\mathcal{L}_2$: Used to prove eligibility for signing a value towards an intended recipient by deriving joint policy fulfillment. This language is used during signing and is depicted below. In the generic scheme, this amounts to proving the ability to decrypt the PE ciphertext in the public key of the recipient.

Theorem 1. *The ul-PCS scheme for generic policies is based on pseudo-random functions, predicate encryption, unforgeable signatures and extractable NIZK systems for languages $\mathcal{L}_1$ and $\mathcal{L}_2$, as defined below. The scheme is T_{Rand} unforgeable, attribute-hiding, and unlinkable, where T_{Rand} is polynomial in the security parameter.*

Formal treatment. The ul-PCS schemes is formally described in Figure 14 using pseudo-code. Furthermore, we introduce a small and obvious helper function for improved readability: the function $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk})$ checks the public-key's well-formedness by verifying the $\mathsf{NIZK}_{\mathcal{L}_1}$ proof of pk and outputs 1 if it verifies, and 0 otherwise. The languages $\mathcal{L}_1$, which guarantees the correctness of the public keys, and the language $\mathcal{L}_2$, which ensures the validity of the signature are defined as follows:

Language $\mathcal{L}_1$: A statement $x_{\mathsf{st}} \coloneqq (T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{vk}^{A}_{\mathsf{sig}}, \mathsf{mpk}_{\mathsf{PE}})$ is in the language $\mathcal{L}_1$ if it holds for a witness $w_{\mathsf{st}} \coloneqq (\mathsf{k}, \mathsf{ctr}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, x, \sigma^1_{\mathsf{sig}}, \sigma^2_{\mathsf{sig}}, \sigma_{\mathsf{ctr}})$ that:
- $\mathsf{ctr} < T_{\mathsf{Rand}}$
- $\mathsf{ct}_{\mathsf{ctr}} = \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, x)$

- $\mathsf{ID}_{\mathsf{ctr}} = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$
- $\mathsf{DS.Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, x), \sigma^1_{\mathsf{sig}}) = 1$ and $\mathsf{DS.Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}), \sigma^2_{\mathsf{sig}}) = 1$
- $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}, (\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ID}_{\mathsf{ctr}}), \sigma_{\mathsf{ctr}}) = 1$

Language $\mathcal{L}_2$: A statement $x_{\mathsf{st}} \coloneqq (\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}^A_{\mathsf{sig}})$ is in the language $\mathcal{L}_2$ if it holds for a witness $w_{\mathsf{st}} \coloneqq (\mathsf{k}, \mathsf{ctr}, \mathsf{sk}_{f_x}, \sigma^3_{\mathsf{sig}})$ that:

- $\mathsf{PE.Dec}(\mathsf{sk}_{f_x}, \mathsf{ct}_R) = 1$
- $\mathsf{ID}_S = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$
- $\mathsf{DS.Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{sk}_{f_x}), \sigma^3_{\mathsf{sig}}) = 1$

The formal theorem and its proof are given in Appendix B.

5.2 ul-PCS for Separable Policies

Identifying predicate encryption as the most heavy tool in this construction, we observe that, for separable policies (cf. Section 2), we can apply a few tricks to remove PE in exchange of ordinary public-key encryption. Towards understanding this section, there are four conceptual items that we are going to replace, which leads to the scheme in Figure 15.

- The functional key sk_{f_x} based on attributes x used to decrypt part of the recipient's public key will be replaced by an ordinary decryption key sk. The reason for that is that we can precompute the matching: the recipient's public key (for, say, attributes y) contains an encryption of $R(y)$ and the sender is given the decryption key if $S(x)$. This mimics the use of a functional key in this case, but at a much lower cost.
- Since the public encryption key can be defined as part of the public parameters, there is no need to sign the corresponding private key sk.
- Signatures on attributes x can be replaced by a signatures on a bit $R(x)$.
- The re-encryption of the PE part of the ciphertext can be replaced by ordinary re-encryptions that admit efficient proofs.

Correspondingly, we also have to change the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ of the scheme, which we modify below. The helper function $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk})$ is defined as in the previous scheme, i.e. it checks the public-key's well-formedness by verifying the $\mathsf{NIZK}_{\mathcal{L}_1}$ proof of pk and outputs 1 if it verifies, and 0 otherwise. Intuitively, this scheme is secure by a reduction to the generic one. In fact, the four replacements above mimic the generic scheme for separable policies. We have:

Theorem 2. *The ul-PCS scheme for separable policies is based on pseudo-random functions, IND-CPA-secure encryption, unforgeable signatures and extractable NIZK systems for languages $\mathcal{L}_1$ and $\mathcal{L}_2$, as defined below. The scheme is T_{Rand} unforgeable, attribute-hiding, and unlinkable, where T_{Rand} is polynomial in the security parameter.*

Formal treatment. The proof of this scheme proceeds analogous to the proof of Theorem 1 since the scheme is a concrete version of the generic scheme. We point out the few minor differences in the proof at the end of Appendices B.2 to B.4. The language $\mathcal{L}_1$, that guarantees the correctness of the public keys, and the language $\mathcal{L}_2$, that ensures the validity of the signature, can be simplified as follows:

$\underline{\mathsf{Setup}(1^\lambda, F)\text{:}}$
$\mathsf{CRS}_{\mathsf{Rand}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Setup}(1^\lambda)$
$\mathsf{CRS}_{\mathsf{Sign}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}^A_{\mathsf{sig}}, \mathsf{vk}^A_{\mathsf{sig}}) \leftarrow \mathsf{DS}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{mpk}_{\mathsf{PE}}, \mathsf{msk}_{\mathsf{PE}}) \leftarrow \mathsf{PE}.\mathsf{Setup}(1^\lambda)$
$\mathsf{mpk} := (T_{\mathsf{Rand}}, F, \mathsf{CRS}_{\mathsf{Rand}}, \mathsf{CRS}_{\mathsf{Sign}}, \mathsf{vk}^A_{\mathsf{sig}}, \mathsf{mpk}_{\mathsf{PE}})$
$\mathsf{msk} := (F, \mathsf{sk}^A_{\mathsf{sig}}, \mathsf{msk}_{\mathsf{PE}})$
Return $(\mathsf{mpk}, \mathsf{msk})$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, x)\text{:}}$
Parse msk as defined above
$\mathsf{k} \leftarrow \{0,1\}^\lambda$
$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{vk}_{\mathsf{sig}}) \leftarrow \mathsf{DS}.\mathsf{Setup}(1^\lambda)$
$\mathsf{sk}_{f_x} \leftarrow \mathsf{PE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{PE}}, f_x)$
$\sigma^1_{\mathsf{sig}} \leftarrow \mathsf{DS}.\mathsf{Sign}(\mathsf{sk}^A_{\mathsf{sig}}, (\mathsf{k}, x)), \sigma^2_{\mathsf{sig}} \leftarrow \mathsf{DS}.\mathsf{Sign}(\mathsf{sk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}))$
$\sigma^3_{\mathsf{sig}} \leftarrow \mathsf{DS}.\mathsf{Sign}(\mathsf{sk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{sk}_{f_x}))$
$\mathsf{usk} := (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, \sigma^1_{\mathsf{sig}}, \sigma^2_{\mathsf{sig}}, \sigma^3_{\mathsf{sig}}, x, \mathsf{sk}_{f_x})$
Return $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{RandKey}(\mathsf{mpk}, (\mathsf{usk}, -1, \perp))$

$\underline{\mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk})\text{:}}$
Parse $\mathsf{mpk}, \mathsf{usk}$ as defined above and $\mathsf{sk} = (\mathsf{usk}, \mathsf{ctr}, \cdot)$
$\mathsf{ctr} := \mathsf{ctr} + 1$
If $\mathsf{ctr} \geq T_{\mathsf{Rand}}$: return $\perp$
$\mathsf{ID}_{\mathsf{ctr}} := \mathsf{PRF}.\mathsf{Eval}(\mathsf{k}, \mathsf{ctr})$
$(\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{ctr}}_{\mathsf{sig}}) \leftarrow \mathsf{DS}.\mathsf{Setup}(1^\lambda)$
$\sigma_{\mathsf{ctr}} \leftarrow \mathsf{DS}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, (\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ID}_{\mathsf{ctr}}))$
$\mathsf{ct}_{\mathsf{ctr}} \leftarrow \mathsf{PE}.\mathsf{Enc}(\mathsf{mpk}_{\mathsf{PE}}, x)$
$\pi_{\mathsf{ctr}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{Rand}}, (T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{vk}^A_{\mathsf{sig}}, \mathsf{mpk}_{\mathsf{PE}}), (\mathsf{usk}, \sigma_{\mathsf{ctr}}))$
$\mathsf{pk}_{\mathsf{ctr}} := (\mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ct}_{\mathsf{ctr}}, \pi_{\mathsf{ctr}})$
Return $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}} := (\mathsf{usk}, \mathsf{ctr}, \mathsf{sk}^{\mathsf{ctr}}_{\mathsf{sig}}))$

Fig. 14a: The setup, key generation and randomization algorithms of the unlinkable PCS scheme for generic policies.

$\underline{\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, \mathsf{pk}_R, m)\text{:}}$
Parse $\mathsf{mpk}, \mathsf{sk} := (\mathsf{usk}, \mathsf{ctr}, \mathsf{sk}_{\mathsf{ctr}})$ and usk as above
If $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_R) = 0$: return $\perp$
$\mathsf{ID}_S := \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$
If $\mathsf{PE.Dec}(\mathsf{sk}_{f_x}, \mathsf{ct}_R) = 0$: return $\perp$
$\pi_s \leftarrow \mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{Sign}}, (\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}^A_{\mathsf{sig}}), \mathsf{sk})$
$\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{ctr}}, (m, \mathsf{pk}_R, \pi_s))$
Return (π_s, σ)

$\underline{\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)\text{:}}$
Parse mpk as defined above and $\sigma = (\pi, \sigma')$
If $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_S) = 0$ or $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_R) = 0$, return $\perp$
Return $(\mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Verify}(\mathsf{CRS}_{\mathsf{Sign}}, (\mathsf{pk}_S, \mathsf{pk}_R), \pi) \ \wedge\ \mathsf{DS.Verify}(\mathsf{vk}_S, (m, \mathsf{pk}_R, \pi), \sigma'))$

Fig. 14b: The signing and verification algorithms of the unlinkable PCS scheme for generic policies.

Language $\mathcal{L}_1$: A statement $x_{\mathsf{st}} := (T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{vk}^{A,R}_{\mathsf{sig}}, \mathsf{pk}^A_{\mathsf{PKE}})$ is in the language $\mathcal{L}_1$ if it holds for a witness $w_{\mathsf{st}} := (\mathsf{k}, \mathsf{ctr}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, m_x, \sigma^1_{\mathsf{sig}}, \sigma_{\mathsf{ctr}})$ that:
- $\mathsf{ctr} < T_{\mathsf{Rand}}$
- $\mathsf{ct}_{\mathsf{ctr}} = \mathsf{PKE.Enc}(\mathsf{pk}^A_{\mathsf{PKE}}, m_x)$
- $\mathsf{ID}_{\mathsf{ctr}} = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$
- $\mathsf{DS.Verify}(\mathsf{vk}^{A,R}_{\mathsf{sig}}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}, m_x), \sigma^1_{\mathsf{sig}}) = 1$
- $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}, (\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ID}_{\mathsf{ctr}}), \sigma_{\mathsf{ctr}}) = 1$

Language $\mathcal{L}_2$: A statement $x_{\mathsf{st}} := (\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}^{A,S}_{\mathsf{sig}}, \mathsf{pk}^A_{\mathsf{PKE}})$ is in the language $\mathcal{L}_2$ if it holds for a witness $w_{\mathsf{st}} := (\mathsf{k}, \mathsf{ctr}, \mathsf{sk}^A_{\mathsf{PKE}}, \sigma^2_{\mathsf{sig}})$ that:
- $\mathsf{PKE.Dec}(\mathsf{sk}^A_{\mathsf{PKE}}, \mathsf{ct}_R) = 1$
- $\mathsf{ID}_S = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$
- $\mathsf{DS.Verify}(\mathsf{vk}^{A,S}_{\mathsf{sig}}, (\mathsf{k}, \mathsf{sk}^A_{\mathsf{PKE}}), \sigma^2_{\mathsf{sig}}) = 1$

5.3 ul-PCS for Role-based Policies

To obtain the scheme for RBAC policies (cf. Section 2) of Figure 16, we are going to "replace" the same four items from the generic scheme, similar to the scheme for separable policies:

- The functional key sk_{f_x} based on attributes x used to decrypt part of the recipient's public key will be replaced by the witness for the accumulator for the roles to which the party can send. This is in general more than one witness, which is more leakage than in the case of PE. This is the reason why we only achieve outsider-security for attribute-hiding in

$\underline{\mathsf{Setup}(1^\lambda, F):}$

$\mathsf{CRS_{Rand}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Setup}(1^\lambda)$

$\mathsf{CRS_{Sign}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Setup}(1^\lambda)$

$(\mathsf{vk}_{\mathsf{sig}}^{A,S}, \mathsf{sk}_{\mathsf{sig}}^{A,S}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$

$(\mathsf{vk}_{\mathsf{sig}}^{A,R}, \mathsf{sk}_{\mathsf{sig}}^{A,R}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$

$(\mathsf{pk}_{\mathsf{PKE}}^{A}, \mathsf{sk}_{\mathsf{PKE}}^{A}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$

$\mathsf{mpk} := (T_{\mathsf{Rand}}, F, \mathsf{CRS_{Rand}}, \mathsf{CRS_{Sign}}, \mathsf{vk}_{\mathsf{sig}}^{A,S}, \mathsf{vk}_{\mathsf{sig}}^{A,R}, \mathsf{pk}_{\mathsf{Enc}}^{A})$

$\mathsf{msk} := (F, \mathsf{sk}_{\mathsf{sig}}^{A,S}, \mathsf{sk}_{\mathsf{sig}}^{A,R}, \mathsf{sk}_{\mathsf{PKE}}^{A})$

Return $(\mathsf{mpk}, \mathsf{msk})$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, x):}$

Parse msk as defined above

$\mathsf{k} \leftarrow \{0,1\}^\lambda$

$(\mathsf{sk_{sig}}, \mathsf{vk_{sig}}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$

$m_x := R(x)$

$\sigma_{\mathsf{sig}}^1 \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{sig}}^{A,R}, (\mathsf{k}, \mathsf{vk_{sig}}, m_x))$

If $S(x) = 1:$

  $\sigma_{\mathsf{sig}}^2 \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{sig}}^{A,S}, (\mathsf{k}, \mathsf{sk}_{\mathsf{PKE}}^{A}))$

  $\mathsf{usk} := (\mathsf{k}, \mathsf{vk_{sig}}, \mathsf{sk_{sig}}, \sigma_{\mathsf{sig}}^1, \sigma_{\mathsf{sig}}^2, m_x, \mathsf{sk}_{\mathsf{PKE}}^{A})$

Else $(S(x) = 0)$:

  $\mathsf{usk} := (\mathsf{k}, \mathsf{vk_{sig}}, \mathsf{sk_{sig}}, \sigma_{\mathsf{sig}}^1, \sigma_{\mathsf{sig}}^2 := \varepsilon, m_x, \mathsf{sk}_{\mathsf{PKE}}^{A} := \varepsilon)$

$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{RandKey}(\mathsf{mpk}, (\mathsf{usk}, -1, \perp))$

Return $(\mathsf{pk}_0, \mathsf{sk}_0)$

Fig. 15a: The setup and key generation algorithm of our unlinkable PCS scheme for separable policies.

$\underline{\mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk}):}$

Parse $\mathsf{mpk}, \mathsf{usk}$ as defined above and $\mathsf{sk} = (\mathsf{usk}, \mathsf{ctr}, \cdot)$

$\mathsf{ctr} := \mathsf{ctr} + 1$

If $\mathsf{ctr} \geq T_{\mathsf{Rand}}$: return $\perp$

$\mathsf{ID}_{\mathsf{ctr}} := \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$

$(\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{sig}}^{\mathsf{ctr}}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$

$\sigma_{\mathsf{ctr}} \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{sig}}, (\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ID}_{\mathsf{ctr}}))$

$\mathsf{ct}_{\mathsf{ctr}} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}^{A}, m_x)$

$\pi_{\mathsf{ctr}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{Rand}}, (T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{A,R}, \mathsf{pk}_{\mathsf{PKE}}^{A}), (\mathsf{usk}, \sigma_{\mathsf{ctr}}))$

$\mathsf{pk}_{\mathsf{ctr}} := (\mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \pi_{\mathsf{ctr}})$

Return $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}} := (\mathsf{usk}, \mathsf{ctr}, \mathsf{sk}_{\mathsf{sig}}^{\mathsf{ctr}}))$

$\underline{\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, \mathsf{pk}_R, m):}$

Parse $\mathsf{mpk}, \mathsf{sk} := (\mathsf{usk}, \mathsf{ctr}, \mathsf{sk}_{\mathsf{ctr}})$ and usk as above

If $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_R) = 0$: return $\perp$

$\mathsf{ID}_S := \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$

If $\mathsf{sk}_{\mathsf{PKE}}^{A} = \varepsilon$: return $\perp$

If $\mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}^{A}, \mathsf{ct}_R) = 0$: return $\perp$

$\pi_s \leftarrow \mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{Sign}}, (\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}_{\mathsf{sig}}^{A,S}, \mathsf{pk}_{\mathsf{PKE}}^{A}), \mathsf{sk})$

$\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{ctr}}, (m, \mathsf{pk}_R, \pi_s))$

Return (π_s, σ)

$\underline{\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma):}$

Parse mpk as defined above and $\sigma = (\pi, \sigma')$

If $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_S) = 0$ or $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_R) = 0$

  Return $\perp$

Return $(\mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Verify}(\mathsf{CRS}_{\mathsf{Sign}}, (\mathsf{pk}_S, \mathsf{pk}_R), \pi) \ \wedge\ \mathsf{DS.Verify}(\mathsf{vk}_S, (m, \mathsf{pk}_R, \pi), \sigma'))$

Fig. 15b: The randomization, signing and verification algorithms of our unlinkable PCS scheme for separable policies.

the case of general matrices. If a party only gets one witness (e.g., in the equality policy), then we achieve full attribute hiding. We further note that the accumulator we need is realized by a weakly-secure short signature scheme [13].
- Accordingly, in this scheme, we sign the witness(es) as opposed to the functional key in the generic scheme.
- Signatures on attributes x needed in the generic scheme do not have a counterpart here. They are not needed by virtue of employing a structure-preserving signature on equivalence-classes (SPS-EQ) that is able to re-randomize public accumulator values while keeping the witnesses intact.
- Re-encryptions can be completely replaced by the re-randomization of SPS-EQ signatures and by relying on the fact that the weakly-sound positive accumulators (constructed from Boneh-Boyen signatures [13]) hide the elements in it.

Correspondingly, we also have to change the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ of this scheme, which we describe below. The helper function for this scheme, $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk})$, is defined differently to the helper function of the previous schemes. In more detail, it verifies the $\mathsf{NIZK}_{\mathcal{L}_1}$ proof as well as the SPS-EQ signature and outputs 1 only if both verifications are successful.

Theorem 3. *The ul-PCS scheme for role-based policies described is based on pseudo-random functions, structure-preserving signatures on equivalence classes, ordinary (unforgeable) signatures, a weakly-sound accumulator, and extractable NIZK systems for languages $\mathcal{L}_1$ and $\mathcal{L}_2$, as defined below. The scheme is T_{Rand} unforgeable, outsider-secure attribute-hiding, and unlinkable, where T_{Rand} is polynomial in the security parameter. It is furthermore T_{Rand}-attribute-hiding for the equality-policy.*

Formal treatment. The proof of this scheme proceeds analogous to the proof of Theorem 1 since the scheme is an optimized version of the generic scheme. We point out the few minor differences at the end of Appendices B.2 to B.4. The corresponding ul-PCS scheme is described in Figure 16. The language $\mathcal{L}_1$ that guarantees the correctness of the public keys, and the language $\mathcal{L}_2$, ensuring the validity of the signature, are defined as follows:

Language $\mathcal{L}_1$: A statement $x_{\mathsf{st}} \coloneqq (T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \vec{M}' \coloneqq (\mathsf{A}_1', \mathsf{A}_2', \mathsf{G}_2'), \mathsf{vk}_{\mathsf{sig}}^{A})$ is in the language $\mathcal{L}_1$ if it holds for a witness $w_{\mathsf{st}} \coloneqq (\mathsf{k}, \mathsf{ctr}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, w_{\mathsf{k}}, \sigma_{\mathsf{sig}}^1, \sigma_{\mathsf{ctr}})$ that:
- $\mathsf{ctr} < T_{\mathsf{Rand}}$
- $\mathsf{ACC.MemVrf}(\mathsf{A}_1', \mathsf{k}, w_{\mathsf{k}}) = 1$ where pp' is defined as $(p, \mathsf{G}_1, \mathsf{G}_2', e)$ (that is, the same as pp but with the generator G_2' instead.)
- $\mathsf{ID}_{\mathsf{ctr}} = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$
- $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}), \sigma_{\mathsf{sig}}^1) = 1$
- $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}, (\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ID}_{\mathsf{ctr}}), \sigma_{\mathsf{ctr}}) = 1$

Language $\mathcal{L}_2$: A statement $x_{\mathsf{st}} \coloneqq (\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}_{\mathsf{sig}}^{A}, \mathsf{pp}', \mathsf{A}')$ is in the language $\mathcal{L}_2$ if it holds for a witness $w_{\mathsf{st}} \coloneqq (\mathsf{k}, \mathsf{ctr}, x, w, \sigma_{\mathsf{sig}}^2)$ that:
- $\mathsf{ACC.MemVrf}(\mathsf{A}', x, w) = 1$
- $\mathsf{ID}_S = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$
- $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A}, (\mathsf{k}, w), \sigma_{\mathsf{sig}}^2) = 1$

$\underline{\mathsf{Setup}(1^\lambda, F):}$

Let pp be a bilinear setup

$\mathsf{CRS}_{\mathsf{Rand}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Setup}(1^\lambda)$

$\mathsf{CRS}_{\mathsf{Sign}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Setup}(1^\lambda)$

$(\mathsf{sk}^A_{\mathsf{sig}}, \mathsf{vk}^A_{\mathsf{sig}}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$

$(\mathsf{vk}^A_{\mathsf{SEQ}}, \mathsf{sk}^A_{\mathsf{SEQ}}) \leftarrow \mathsf{SEQ.KeyGen}_{\mathcal{R}}(\mathsf{pp})$

Parse F as an RBAC matrix with

n_R roles denoted by $1, \ldots, n_R$

For all $y \in [n_R]$:

  $(\mathsf{A}_y, \alpha_y) \leftarrow \mathsf{ACC.Create}(\mathsf{pp})$

  $S_y \leftarrow \{i \in [n_R] : F(i, y) = 1\}; W_y \leftarrow ()$

For all $i \in S_y$:

  $w_i \leftarrow \mathsf{ACC.Add}(\mathsf{pp}, \mathsf{A}_y, \alpha_y, i)$

  $W_y \leftarrow W_y || (i, w_i)$

$\mathsf{CRS} := (\mathsf{CRS}_{\mathsf{Rand}}, \mathsf{CRS}_{\mathsf{Sign}})$

$\mathsf{mpk} := (\mathsf{pp}, T_{\mathsf{Rand}}, F, \mathsf{CRS}, \mathsf{vk}^A_{\mathsf{sig}}, \mathsf{vk}^A_{\mathsf{SEQ}})$

$\mathsf{msk} := (\mathsf{pp}, F, \mathsf{sk}^A_{\mathsf{SEQ}}, (\mathsf{A}_j, W_j)_{j=1}^{n_R})$

Return $(\mathsf{mpk}, \mathsf{msk})$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, x):}$

Parse msk as defined above

$\mathsf{k} \leftarrow \{0,1\}^\lambda$

$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{vk}_{\mathsf{sig}}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$

$(\mathsf{A}_{\mathsf{k}}, \alpha_{\mathsf{k}}) \leftarrow \mathsf{ACC.Create}(\mathsf{pp})$

$w_{\mathsf{k}} \leftarrow \mathsf{ACC.Add}(\mathsf{A}_{\mathsf{k}}, \alpha_{\mathsf{k}}, \mathsf{k})$

$\vec{M} := (\mathsf{A}_{\mathsf{k}}, \mathsf{A}_x, \mathsf{G}_2)$

$\sigma_{\mathsf{SEQ}} \leftarrow \mathsf{SEQ.Sign}_{\mathcal{R}}(\mathsf{sk}^A_{\mathsf{SEQ}}, \vec{M})$

$\sigma_{\mathsf{sig}} \leftarrow \mathsf{DS.Sign}(\mathsf{sk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}))$

$W := ()$

For each $y \in [n_R] : F(x, y) = 1$ do:

  Retrieve $(x, w) \in W_y$

  $W \leftarrow W || (w, \mathsf{DS.Sign}(\mathsf{sk}^A_{\mathsf{sig}}, (\mathsf{k}, w)))$

$\mathsf{usk} = (\vec{M}, \sigma_{\mathsf{SEQ}}, W, w_{\mathsf{k}}, \mathsf{k}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, \sigma_{\mathsf{sig}}, x)$

$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{RandKey}(\mathsf{mpk}, (\mathsf{usk}, -1, \perp))$

Return $(\mathsf{pk}_0, \mathsf{sk}_0)$

Fig. 16a: The setup and key generation algorithm of our unlinkable PCS scheme for RBAC policies.

$\underline{\mathsf{RandKey}(\mathsf{mpk}, \mathsf{sk}):}$

Parse $\mathsf{mpk}, \mathsf{usk}$ as defined above and $\mathsf{sk} = (\mathsf{usk}, \mathsf{ctr}, \cdot)$

$\mathsf{ctr} \coloneqq \mathsf{ctr} + 1$

If $\mathsf{ctr} \geq T_{\mathsf{Rand}}$: return $\perp$

$\mathsf{ID}_{\mathsf{ctr}} \coloneqq \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$

$(\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{ctr}}_{\mathsf{sig}}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$

$\sigma_{\mathsf{ctr}} \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{sig}}, (\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ID}_{\mathsf{ctr}}))$

$\mu_{\mathsf{ctr}} \leftarrow \mathbb{Z}_p^*$

$(\vec{M}', \sigma'_{\mathsf{SEQ}}) \leftarrow \mathsf{SEQ.ChgRep}_{\mathcal{R}}(\mathsf{vk}^A_{\mathsf{SEQ}}, \vec{M}, \sigma_{\mathsf{SEQ}}, \mu_{\mathsf{ctr}})$

$\pi_{\mathsf{ctr}} \leftarrow \mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{Rand}}, (T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \vec{M}', \mathsf{vk}^A_{\mathsf{sig}},), (\mathsf{usk}, \sigma_{\mathsf{ctr}}))$

$\mathsf{pk}_{\mathsf{ctr}} \coloneqq (\mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \vec{M}', \sigma'_{\mathsf{SEQ}}, \pi_{\mathsf{ctr}})$

Return $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}} \coloneqq (\mathsf{usk}, \mathsf{ctr}, \mathsf{sk}^{\mathsf{ctr}}_{\mathsf{sig}}))$

$\underline{\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, \mathsf{pk}_R, m):}$

Parse $\mathsf{mpk}, \mathsf{sk} \coloneqq (\mathsf{usk}, \mathsf{ctr}, \mathsf{sk}_{\mathsf{ctr}})$ and usk as above

If $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_R) = 0$: return $\perp$

$\mathsf{ID}_S \coloneqq \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})$

Parse $\mathsf{pk}_R = (\ldots, (\mathsf{A}, \mathsf{A}', \mathsf{G}'_2), \ldots)$

Let $\mathsf{pp}' \leftarrow (p, \mathsf{G}_1, \mathsf{G}'_2, e)$

If $\nexists\ w^* \in W \mid \mathsf{ACC.MemVrf}(\mathsf{pp}', \mathsf{A}', x, w^*)$: return $\perp$

Find $w^* \in W \mid \mathsf{ACC.MemVrf}(\mathsf{pp}', \mathsf{A}', x, w^*) = 1$

$\pi_s \leftarrow \mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Prove}(\mathsf{CRS}_{\mathsf{Sign}}, (\mathsf{ID}_S, \mathsf{vk}^A_{\mathsf{sig}}, \mathsf{pp}', \mathsf{A}'), \mathsf{sk})$

$\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}_{\mathsf{ctr}}, (m, \mathsf{pk}_R, \pi_s))$

Return (π_s, σ)

$\underline{\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S, \mathsf{pk}_R, m, \sigma):}$

Parse mpk as defined above and $\sigma = (\pi, \sigma')$

If $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_S) = 0$ or $\mathsf{ValidPK}(\mathsf{mpk}, \mathsf{pk}_R) = 0$

  Return $\perp$

Return $(\mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Verify}(\mathsf{CRS}_{\mathsf{Sign}}, (\mathsf{pk}_S, \mathsf{pk}_R), \pi) \ \wedge \ \mathsf{DS.Verify}(\mathsf{vk}_S, (m, \mathsf{pk}_R, \pi), \sigma'))$

Fig. 16b: The rerandomization, signing and verification algorithm of our unlinkable PCS scheme for RBAC policies.

6 Instantiation and Performance Analysis

Before discussing the performance of our proposed schemes, we give a brief overview of the concrete instantiations of the cryptographic primitives. The full documentation on how the different primitives are used to realize our schemes for the various policies can be found in Appendices C and D.

6.1 Overview of Cryptographic Algorithms and Proof Systems

Digital Signatures. We require three different types of signature schemes to instantiate the proposed constructions. We use BLS signatures [15] wherever appropriate. For SPS-EQ we use the construction proposed in [37]. Additionally, if we need a Groth-Sahai (GS) friendly relation (see below), we use the structure-preserving signatures (SPS)—for example when users need to prove the knowledge of hidden messages and signatures that successfully verify under the verification key of the CA. For simplicity, we utilize a slightly modified variant of FHS's SPS-EQ [37], without the change representation algorithm, as our implementation of standard SPS.

Predicate Encryption. The proposed generic ul-PCS scheme in Figure 14 relies on PE. We use the Okamoto-Takashima (OT12) [55] scheme based on dual pairing vector spaces that realizes the inner-product predicate functionality.

Public-Key Encryption. The proposed ul-PCS scheme with separable policies relies on public key encryptions for which we use ElGamal encryption [30] as an instantiation.

Pseudorandom Functions. We utilize the Dodis-Yampolskiy PRF [29] as a well-known and efficient PRF that operates over a cyclic group $\mathbb{G}$ of prime order p.

Non-Interactive Zero-Knowledge. Combining all of these concrete instantiations allows us to formalize the NP-relations described in the proposed constructions. We rely on three well-known proof systems: Sigma protocols, Groth-Sahai proofs, and range-proofs. We give a brief overview and defer the details to Appendix C.

We use the standard sigma protocols [58] as well as some recent techniques described in [28, 51]. To make the schemes non-interactive, we use the Fiat-Shamir paradigm [35] w.r.t. a hash functions $H : \{0,1\}^* \to \mathbb{Z}_p$. In essence, whenever the prover has the knowledge of scalar witnesses, we use sigma protocols to obtain an efficient zero-knowledge proof. Additionally, we use Groth-Sahai (GS) proof systems [45] when the witness is a group element with hidden discrete logarithms. Over an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \mathsf{G}_1, \mathsf{G}_2)$, this construction can prove the satisfiability of any pairing-product equation (PPE) of the form $\prod_{i=1}^{n} e(A_i, \mathcal{Y}_i) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{j=1}^{m} \prod_{i=1}^{n} e(\mathcal{X}_j, \mathcal{Y}_i)^{\gamma_{i,j}} = T$, where $\mathcal{X}_1, \ldots, \mathcal{X}_m \in \mathbb{G}_1$, $\mathcal{Y}_1, \ldots, \mathcal{Y}_n \in \mathbb{G}_2$ are the witnesses given as a commitment and $T \in \mathbb{G}_T$, $A_1, \ldots, A_n \in \mathbb{G}_1$, $B_1, \ldots, B_m \in \mathbb{G}_2$ and $\{\gamma_{i,j}\}_{i \in [1,m], j \in [1,n]} \in \mathbb{Z}_p$ are constant values which are a part of the instance or publicly known. Another advantage of using GS proofs is the ability to use verification batching techniques, such as the one described in this paper [47]. Finally, the

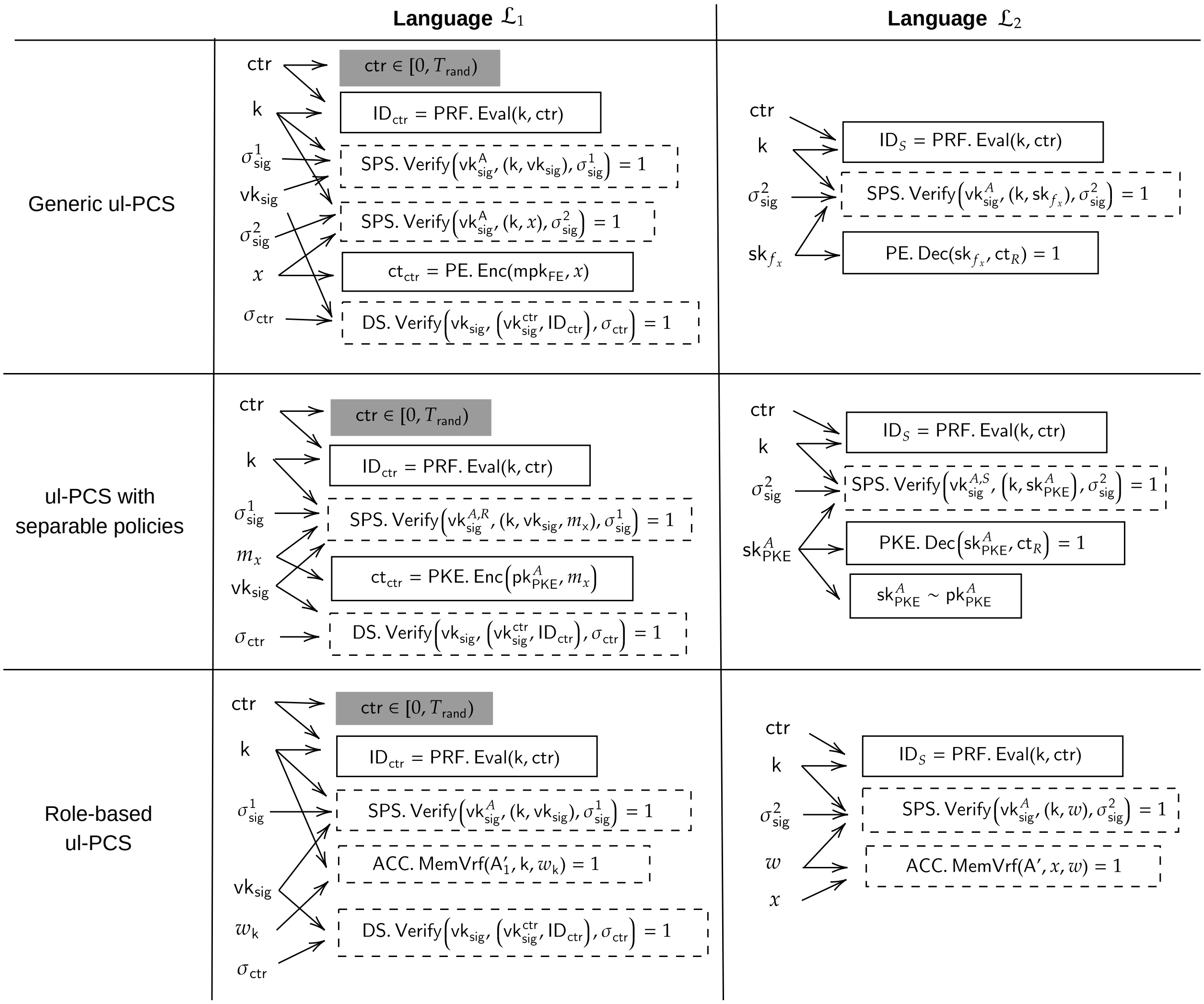

Fig. 17: NP-relations and witnesses for our ul-PCS constructions and the used proof systems: Sigma protocols, range-proofs and Groth-Sahai proofs. For the sake of concreteness, we use the notation SPS to indicate where we need structure-preserving signatures.

range-proof allows a user to prove that a hidden value lies within a certain range. In the proposed constructions, the number of re-randomizations is upper bounded by a maximum number T_{Rand} that is fixed in the setup. In order to prove that this condition is fulfilled, our instantiation relies on range-proofs proposed by Bünz et al. [17], known as Bulletproofs. We summarize in Figure 17 all the relations and proof systems used in our constructions. The concrete realizations of the NIZK relations are detailed in Appendix D.

6.2 Performance Analysis

Benchmark & Environment. We implement the proposed ul-PCS schemes and evaluate their performance based on BN-254 elliptic curve groups [9], $y^2 = x^3 + b$, with embedding curve degree 12, where the first group $\mathbb{G}_1$ is a standard curve defined over $\mathbb{Z}_p$. The second group

Table 1: Cost of basic operations. $\mathbf{M}_i$ and $\mathbf{E}_i$ denote multiplication and exponentiation costs in $\mathbb{G}_i$ for $i \in \{1, 2, T\}$, respectively. $\mathbf{P}$ denotes pairing cost. ms stands for millisecond and μs for microsecond.

Elliptic-Curve	Library	$\mathbf{M}_1$ time (μs)	$\mathbf{E}_1$ time (ms)	$\mathbf{M}_2$ time (μs)	$\mathbf{E}_2$ time (ms)	$\mathbf{M}_T$ time (μs)	$\mathbf{E}_T$ time (ms)	$\mathbf{P}$ time (ms)
BN-254	Charm-Crypto	3.3	0.9	7.1	1.6	21.4	4.8	18.5
BN-256	bplib	3.8	0.3	6	1	3	2.3	2.74

$\mathbb{G}_2$ and target group $\mathbb{G}_T$ are defined over the extension fields $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_{p^{12}}$, respectively. We use the Charm-crypto framework [2] written in Python as the main framework and our open-source implementation is available at [61]. In our experiments, we used a machine that we believe represents typical workloads for ul-PCS. We used an HP Zbook 15 G6 with 16 GB of RAM, an Intel Core i7-9850H CPU @ 2.60GHz, and an SSD for storage, running Ubuntu 22.04 LTS.

Remark 3. It is essential to emphasize that the provided evaluation should be viewed as an academic prototype. Its primary purpose is to establish an initial estimate of complexity and is not intended to meet production-grade standards. This is primarily due to the significant impact that the choice of an elliptic curve and the underlying library has on the security and performance of our proposed constructions. Table 1 lists the average runtime based on 1000 independent measurements for the basic operations in the BN-254 curve using the Charm-Crypto framework. For the purpose of comparison, we have included the running time obtained from the bplib[7] library over BN-256 pairing-friendly curves. Bplib relies on OpenPairing[8] which utilizes OpenSSL as the underlying arithmetic framework for implementing an efficient bilinear pairing over the BN curve. It offers promising indications that a production-ready implementation could achieve significantly faster performance, possibly by an order of magnitude. It's worth noting that even a simple switch from the EC library from PBC to OpenPairing results in a minimum of 6 times faster pairing operations and, since our schemes heavily rely on such operations, this improvement is particularly noteworthy. However, due to the broader adoption and smoother code integration offered by Charm-Crypto, our primary intention to use it was for the implementation of our proof of concept.

Optimized constructions for special policies. All computations are performed on the same machine and we achieve practical results, even for large attribute sets and policies. We report the execution times on an average of 100 executions without preprocessing. The maximum number of re-randomizations is assumed to be $T_{\mathsf{Rand}} = 2^{16} - 1$. The length of secret key

[7] https://github.com/gdanezis/bplib.
[8] https://github.com/dfaranha/OpenPairing.

Table 2: Running time of constant operations and size of constant parameters.

Scheme	KeyGen time (ms)	RandKey time (ms)	Verify time (ms)	pk size (kbytes)	σ size (kbytes)
ul-PCS, role-based	750	550	1 630	28	16
ul-PCS, separable policies	490	480	1 020	28	14.5

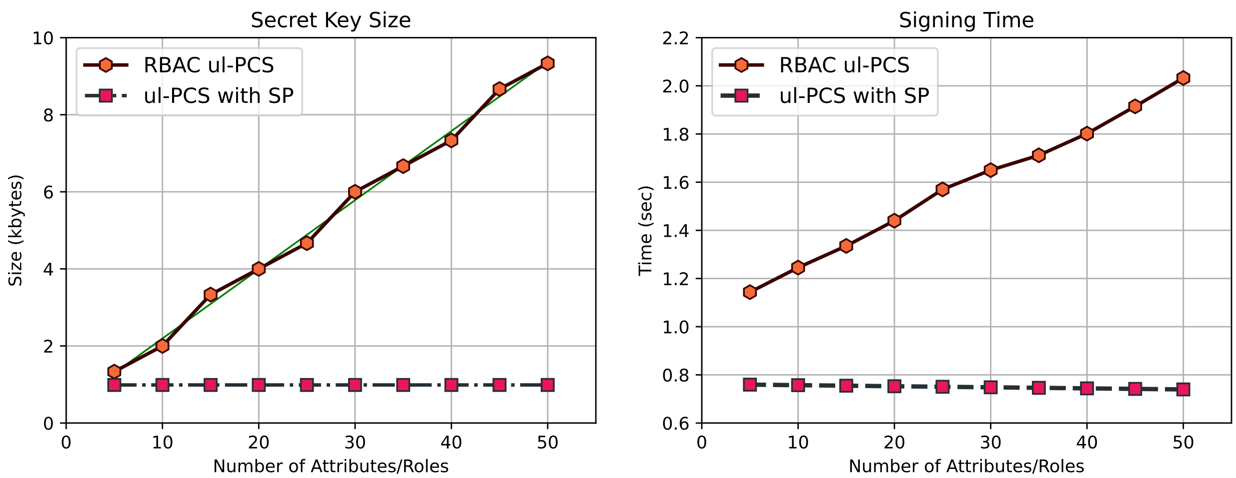

Fig. 18: Secret-key size and Signing time of our RBAC ul-PCS and ul-PCS for separable policies. Left: secret-key size versus the number of attributes/roles; Right: signing time versus the number of attributes/roles.

and signing time are shown in Figure 18. Table 2 depicts the constant execution times and parameter sizes. A summary of our analysis is as follows:

In the role-based ul-PCS described in Figure 16, it takes around 1.2 seconds (resp. 2 seconds) to sign a message using a secret key with 5 roles (resp. 50 roles). The average cost per additional attribute is around 19 ms. It takes around 1.6 seconds for a verifier to verify the validity of a signature independent of the number of roles. The required memory for a user to store a secret key representing 5 roles (resp. 50 roles) is not larger than 2 kbytes (resp. 10 kbytes) and we pay 270 bytes per additional attribute. The corresponding public key has a constant size of 28 kbytes, again, independent of the number of roles. A signature in this scheme has a constant size of 16 kbytes.

The proposed ul-PCS scheme for separable policies described in Figure 15, achieves a slightly better performance in most operations. More precisely, a secret key can be generated in less than 490 ms independent of the number of attributes. The key re-randomization phase also benefits from this constant running time and requires 480 ms to be executed. Unsurprisingly, as illustrated in Figure 18, the signing time is also constant and it takes around 750 ms to sign a message. Signature verification takes around 1 second. The secret

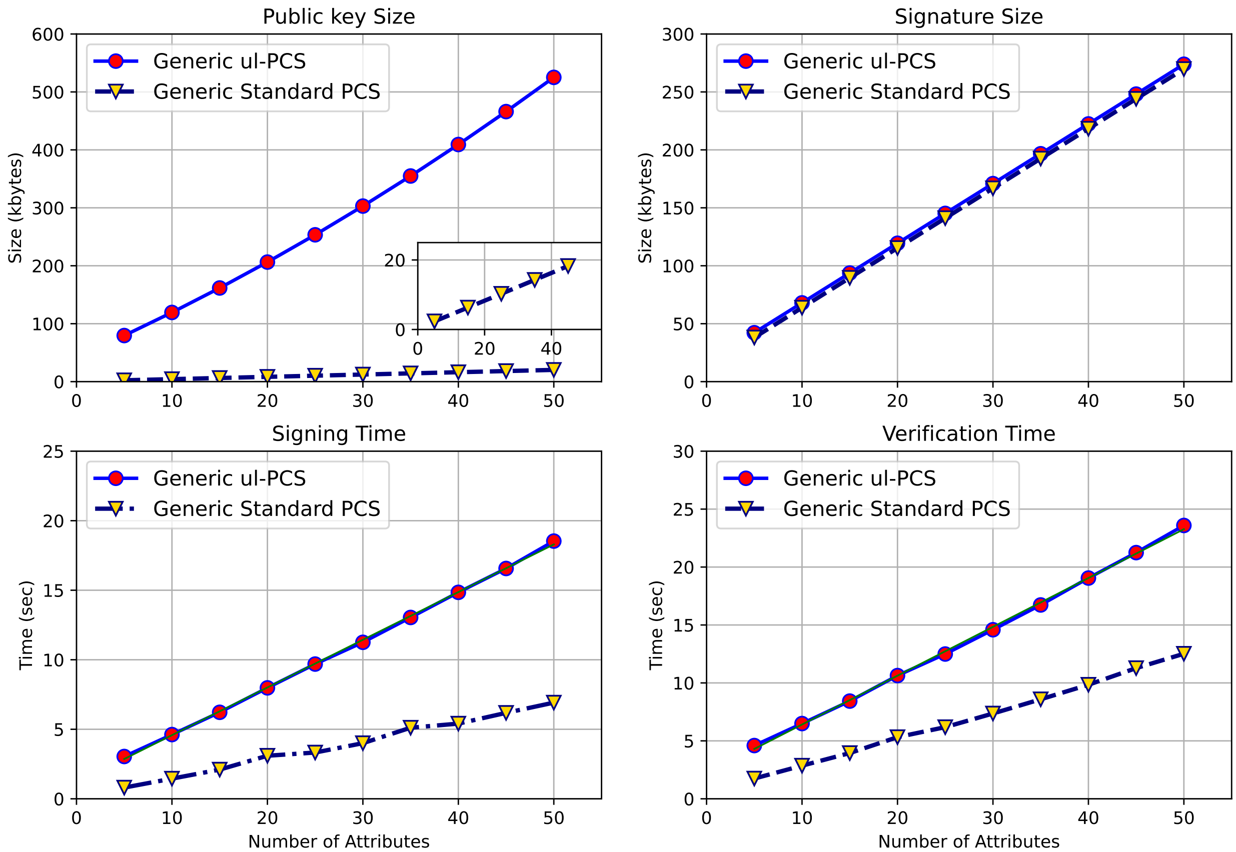

Fig. 19: Performance of the generic ul-PCS and standard PCS schemes. The x-axis denotes the number of attributes. Top left: public-key sizes; Top-Right: signature sizes; Bottom left: signing times; Bottom right: verifying times.

keys are also constant and require only 1 kbyte of storage while the corresponding public key is 28 kbytes large. A signature has a constant size of 14.5 kbytes, which is slightly shorter than for the role-based ul-PCS.

Generic ul-PCS/PCS with IP-PE. In Figure 19, we compare the overhead of the proposed generic ul-PCS scheme with the standard PCS scheme proposed by Badertscher et al. [6]. Here, we use the same inner-product predicate encryption scheme for both prototypes. We are furthermore interested in the dependency on the number of attributes, where attributes are encoded as length-n vectors (over a base field). Due to space constraints, we focus our attention on the "online operations" that are part of both schemes (signing and verification times; signature and public key sizes). We explore the range between 5 and 50 attributes. We observe that standard PCS generally has better performance characteristics, which can be attributed to the cost of unlinkability/anonymity.

A public key for 5 attributes in the generic ul-PCS scheme has a size of around 79 kbytes and we pay 9.9 kbytes per additional attribute. In the generic standard PCS scheme we start

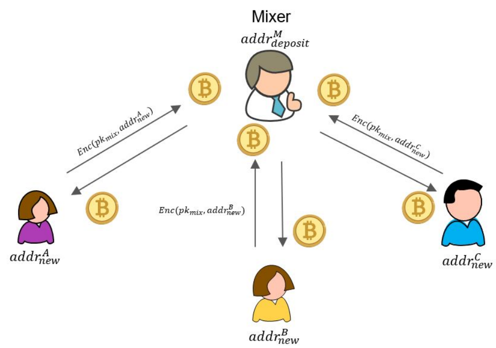

Fig. 20: A simplified illustration of the main idea behind a mixer: parties generate new addresses which they communicate to the mixer in a secure way. The mixer accumulates coins at the deposit address and redistributes them when the number of participants is exceeding a threshold (which defines the anonymity set).

at 2.3 kbytes and pay 390 bytes per additional attribute. Signature sizes are almost identical in both schemes, a signature generated in the case of 5 attributes is around 40 kbytes and each additional attribute incurs a cost of about 5.14 kbytes. In terms of signing times, in the generic ul-PCS a message can be signed in around 3 seconds (resp. 18.5 seconds) in case of 5 attributes (resp. 50 attributes). The average cost per additional attribute in this range from 5 to 50 attributes can be estimated with 340 ms. For standard PCS, we are require about 800 ms (resp. up to 6.9 seconds) to generate a signature for 5 attributes (resp. 50 attributes). The average cost per additional attribute here is 130 ms. Finally, verification of a signature in the context of 5 attributes (resp. 50 attributes) takes about 4.59 seconds (resp. 23 seconds) for the generic ul-PCS scheme. For the standard PCS scheme the verification time is around 1.74 seconds (resp. 12.5 seconds) for 5 (resp. 50) attributes. In this range, the price per additional attribute can be estimated at 420 ms for the generic ul-PCS scheme and 230 ms for the standard PCS scheme.

7 Application to Payment Systems

We discuss the application of ul-PCS to pseudonymous UTxO-based transaction systems like Bitcoin, and to privacy-preserving transaction systems like Zcash [11] and Monero [3]. Furthermore, we also discuss how the enrollment process can be implemented in a distributed manner to avoid a single point of failure. In the last step, we explain how ul-PCS can enrich centralized currencies such as CBDCs.

7.1 Integration with UTxO-based Systems and Compliant Mixing

In UTxO-based systems like Bitcoin [53], a data structure of unspent transaction outputs is maintained by the ledger, where an unspent transaction output can be thought of as an address-value pair. In principle, every address corresponds to a public key (in Bitcoin it is the hash of a public key), and the value sitting at the address can be spent if a signature can be generated (signing a particular transaction spending the value) that verifies with respect to that public key. With respect to privacy, this system is pseudonymous: anyone can link transactions, but the real-world entity behind a key is not deducible from the ledger. To obfuscate the transaction graph, several techniques have been proposed for Bitcoin [38,52,60], all of them requiring the ability that a party can generate fresh public keys (i.e., addresses) at will. For example, in a mixer solution like Obscuro [60], parties would communicate privately a new return address to the mixer, send coins, and the mixer sends back the coins to the shuffled return addresses of many users.

Following [6], a ul-PCS scheme can naturally be coupled with such transaction systems to achieve policy-compliance with strong privacy guarantees (hiding the attributes of parties) and without the need for privacy revocation, due to the cryptographic enforcement of the policy. However, the original PCS scheme did not allow a real-world entity to generate new public keys on their own, which makes them incompatible with the privacy-enhancing techniques above. Unlinkable PCS on the other hand enables all obfuscation techniques based on freshly generated addresses while delivering all guarantees of a PCS scheme which makes it an attractive solution for policy enforcement at the transaction level. Unlinkable PCS cryptographically enforces a policy and enables a party to generate unlinkable public keys (i.e., addresses) that are all connected to its attributes. This opens the door for mixing services for UTxO based ledgers (such as Obscuro) to be provably compliant without revoking the privacy at any time: by equipping accredited users and mixers with appropriate credentials (connected to their jurisdiction), one can enforce on a transaction level that the service only serves admissible customers.

7.2 Integration with DAP Systems

Overview. We now show how unlinkable policy-compliant signatures can be formally integrated with decentralized anonymous payment (DAP) systems such as Monero [3] or Zcash [11]. In a recent work, Engelmann et al. [31] introduced a new abstraction model for protocols like Zcash and Monero to reason about their confidentiality, privacy, and soundness properties. Recall that in such UTxO-like privacy-preserving transaction systems, confidentiality and anonymity of transacted amounts and involved addresses must be ensured. For example, the amount, or more generally speaking the output of a transaction, could be encrypted using the public key of the recipient and, by using key-private encryption [11], outputs transactions which are unlinkable when posted on a public ledger. To enable the UTxO functionality, such privacy-preserving UTxO like schemes must allow the generation of what is often called a "nullifier" value that anonymously marks a transaction output as spent.

To abstract the concrete mechanisms used in different systems, Engelmann et al. [31] define the notion of a one-time account (OTA) scheme. We recall the definition of an OTA scheme in Appendix F, which is a tuple of algorithms $\mathsf{OTA} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{NoteGen}, \mathsf{Enc}, \mathsf{Receive}, \mathsf{NulEval})$. An OTA scheme is the privacy-preserving analogue and generalization of a plain UTxO based transaction system described above: instead of connecting a transaction output to a long-term cryptographic key (e.g. by including a hash of the recipient's public key as part of the output), an OTA scheme allows to generate (knowing the intended recipients' public keys), for each transaction output, a unique and anonymous one-time account which is called a "note". The contents of a note can only be accessed using the recipient's public key, which can be further used to claim it by computing a unique nullifier value that anonymously marks it as spent. If the intended recipient requires auxiliary information to create the nullifier, an OTA scheme has an explicit function to encrypt such values towards the recipient (this ciphertext is formally part of the transaction output and accompanies the note).

Based on an OTA scheme, the main task of a higher-level transaction system (such as Zcash or Monero) is to maintain a ledger, recording notes in its state, whether they have been spent or not, and to implement a certain monetary policy (such as conservation of money during a standard transaction, or how to mint coins in special transactions). This is highly application dependent, and the OTA scheme provides the core infrastructure underneath. The high-level transaction mechanic is as follows: in a transaction, one declares knowledge of (input) notes contained in the ledger state and presents their nullifiers plus a NIZK proof that they are constructed correctly based on the input notes. Importantly, the transaction reveals no link to the input notes other than their containment in the ledger state (the nullifiers ensure that no note can be spent more than once). Finally, a transaction specifies a new set of output notes, and an application-dependent proof that the output notes stand in a particular relation with the input notes (such as that the sum of all inputs equals the sum of all outputs minus a given fee). We refer to [11] and [31] on how these systems can be constructed based on an OTA scheme.

We now present two constructions how to combine PCS with OTA to achieve accounts that are bundled with private attributes about which policy compliance can be proved. The first construction is the generic composition of PCS and OTA, while the second construction constitutes an efficiency improvement in case the PCS is unlinkable.

Construction I. The idea is to use the PCS scheme to sign the note but hide the involved public keys and signatures inside the OTA ciphertext. The owner of a note can then use the relevant values in a zero-knowledge proof of knowledge when claiming, as described above, a note as part of a transaction.

This generic composition is simple and obviously preserves all underlying OTA guarantees (cf. Appendix F), but pushes a lot of complexity into the NIZK. In order to show that the nullifier nul spends a note note (which contains a vector of type-value pairs $\vec{a}$ and is generated with randomness r see Appendix F) that is contained in the ledger state st, at least the following language must be supported for the construction:

$$\begin{aligned} L = \{ &(\mathsf{mpk}, st, \mathsf{nul}) \mid \exists (\mathsf{note}, \mathsf{sk}_{\mathrm{ota}}, \vec{a}, r, \mathsf{pk}^S_{\mathrm{pcs}}, \mathsf{pk}^R_{\mathrm{pcs}}, \mathsf{sk}^R_{\mathrm{pcs}}, \sigma_{\mathrm{pcs}}) : \\ &\mathsf{note} \in st \wedge \mathsf{note} = \mathsf{NoteGen}(P(\mathsf{sk}_{\mathrm{ota}}), \vec{a}, r) \wedge \mathsf{nul} = \mathsf{NulEval}(\mathsf{sk}_{\mathrm{ota}}, r) \wedge \\ &\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}^S_{\mathrm{pcs}}, pk^R_{\mathrm{pcs}}, \mathsf{note}, \sigma_{\mathsf{pcs}}) \wedge \mathsf{pk}^R_{\mathrm{pcs}} = P(\mathsf{sk}^R_{\mathrm{pcs}}) \}, \end{aligned}$$

where $P(\mathsf{sk})$ is an assumed mapping that computes the public key from the secret key (e.g., to prove knowledge of the secret key).

Construction II. Now, we present a much more efficient way to compose the two schemes while retaining essentially the same privacy guarantees as the first construction above. We obtain the mentioned efficiency gains by leveraging the unlinkability feature of the ul-PCS scheme. First, we describe the scheme and argue about its security below. The scheme works as follows: the sender creates a note according to the OTA scheme and PCS-signs a commitment to the note such that it verifies with the sender's and recipient's current PCS public keys respectively. We leave the format of the note unchanged and transmit the additional information as well as the commitment opening as part of the ciphertext of the OTA scheme. The nullifier on the other hand will be the OTA nullifier, both PCS public keys, the PCS signature on the commitment, plus another PCS signature created by the recipient on the OTA nullifier, and a NIZK that proves knowledge of the opening information of the commitment. (Recall that PCS-Signing requires specifying a target public key which is not relevant at this step. For simplicity, we assume that a party can "sign towards itself", in which case standard signatures are a special case of PCS.) In summary, a party can only claim ownership of a note (by constructing the nullifier) if it possesses the underlying OTA private key (to decrypt the output and to generate the OTA nullifier) *and* possesses the PCS private key that corresponds to the PCS public key towards which the note was created, i.e., for which the signature on the note successfully verifies. We observe that this construction avoids a NIZK about PCS signatures and gets away with just a simple commitment proof. The scheme is formally given below.

We note that the only change to the interface is that KeyGen can have black-box access to a PCS key-gen oracle and that it is parameterized by an attribute x. Thanks to this modularity, the OTA security requirements remain well-defined.

Setup: Run $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{ul\text{-}PCS.Setup}$ and $\mathsf{pp} \leftarrow \mathsf{OTA.Setup}$ and define the public parameter $\mathsf{p} \leftarrow (\mathsf{mpk}, \mathsf{pp})$. For simplicity, p is implicitly provided to all algorithms below and not explicitly mentioned.

$\mathsf{KeyGen}_x^{\mathsf{KeyGen}_{\mathrm{PCS}}(\mathsf{msk},\cdot)}$: Run $(\mathsf{pk}_{\mathrm{ota}}, \mathsf{sk}_{\mathrm{ota}}) \leftarrow \mathsf{OTA.KeyGen}$ and obtain $(\mathsf{pk}_{\mathrm{pcs}}, \mathsf{sk}_{\mathrm{pcs}})$ for attribute x via an oralce call. Define $\mathsf{pk} = (\mathsf{pk}_{\mathrm{ota}}, \mathsf{pk}_{\mathrm{pcs}})$ and $\mathsf{sk} = (\mathsf{sk}_{\mathrm{ota}}, \mathsf{sk}_{\mathrm{pcs}})$.

$\mathsf{NoteGen}((\mathsf{pk}^R_{\mathrm{ota}}, \mathsf{pk}^R_{\mathrm{pcs}}), \vec{a}, (r_1, r_2))$: Create $\mathsf{note} \leftarrow \mathsf{OTA.NoteGen}(\mathsf{pk}_{\mathrm{ota}}, \vec{a}, r_1)$.

$\mathsf{Enc}((\mathsf{pk}^R_{\mathrm{ota}}, \mathsf{pk}^R_{\mathrm{pcs}}), \vec{a}, (r_1, r_2), (\mathsf{sk}_{\mathrm{ota}}, \mathsf{sk}_{\mathrm{pcs}}), \xi)$: Re-create the note note using r_1 as above and compute $\mathsf{Com} \leftarrow \mathsf{Commit}(\mathsf{note}; r_2)$. Run $(\mathsf{sk}'_{\mathrm{pcs}}, \mathsf{pk}'_{\mathrm{pcs}}) \leftarrow \mathsf{ul\text{-}PCS.RandKey}(\mathsf{sk}_{\mathrm{pcs}})$ and store the new PCS keys. Run $\sigma_{\mathrm{note}} \leftarrow \mathsf{ul\text{-}PCS.Sign}(\mathsf{sk}'_{\mathrm{pcs}}, \mathsf{pk}^R_{\mathrm{pcs}}, \mathsf{Com})$. Compute $C \leftarrow \mathsf{Enc}\left(\mathsf{pk}^R_{\mathrm{ota}}, (\vec{a}, (r_1, r_2), \mathsf{pk}'_{\mathrm{pcs}}, \mathsf{pk}^R_{\mathrm{pcs}}, \sigma_{\mathrm{note}}), \xi\right)$.

$\mathsf{Receive}(\mathsf{note}, C, (\mathsf{sk}_{\mathrm{ota}}, \mathsf{sk}_{\mathrm{pcs}}))$: Compute $\mathsf{OTA.Receive}(\mathsf{note}, C, \mathsf{sk}_{\mathrm{ota}})$.

$\mathsf{NulEval}((\mathsf{sk}^R_{\mathrm{ota}}, \mathsf{sk}^R_{\mathrm{pcs}}), \vec{a}, (r_1, r_2), \mathsf{pk}^S_{\mathrm{pcs}}, \mathsf{pk}^R_{\mathrm{pcs}}, \sigma_{\mathrm{note}})$: Verify that $\mathsf{pk}^R_{\mathrm{pcs}}$ is the public key corresponding to $\mathsf{sk}^R_{\mathrm{pcs}}$ (otherwise, abort). Generate $\mathsf{nul}' \leftarrow \mathsf{OTA.NulEval}(\mathsf{sk}, r_1)$, compute $\sigma_{\mathrm{nul}} \leftarrow \mathsf{ul\text{-}PCS.Sign}(\mathsf{sk}^R, \mathsf{pk}^R, \mathsf{nul}')$, and recreate the commitment Com (using $\vec{a}$, r_1, and r_2). Check that $\mathsf{ul\text{-}PCS.Verify}(\mathsf{pk}^S_{\mathrm{pcs}}, \mathsf{pk}^R_{\mathrm{pcs}}, \mathsf{Com}, \sigma_{\mathrm{note}}) = 1$ (otherwise abort). Finally, output $\mathsf{nul} \leftarrow (\mathsf{nul}', \mathsf{Com}, \mathsf{pk}^S_{\mathrm{pcs}}, \mathsf{pk}^R_{\mathrm{pcs}}, \sigma_{\mathrm{note}}, \sigma_{\mathrm{nul}})$.

For this scheme, we require a NIZK for the following language, which is known to admit efficient proof systems [31, Section 5]:

$$L' = \{(st, \mathsf{Com}, \mathsf{nul}') \,|\, \exists(\mathsf{note}, \mathsf{sk}_{\mathrm{ota}}, \vec{a}, r_1, r_2) : \mathsf{note} \in st \,\wedge \mathsf{Com} = \mathsf{Commit}(\mathsf{note}, r_2) \\ \wedge\, \mathsf{note} = \mathsf{NoteGen}(P(\mathsf{sk}_{\mathrm{ota}}), \vec{a}, r_1) \,\wedge \mathsf{nul}' = \mathsf{NulEval}(\mathsf{sk}_{\mathrm{ota}}, r_1)\}.$$

Security analysis. We now elaborate on the guarantees provided by our construction. While it is easy to see that the strawman approach is as secure as OTA, for the more efficient construction above, we trade some security for efficiency. Next, we elaborate on the security provided by that construction following the OTA security goals.

Soundness and binding. An OTA ciphertext should decrypt to values that would correctly reconstruct the note that was given to it. On the other hand, binding ensures that a note is essentially a binding commitment to the vector $\vec{a}$. Both of these properties are satisfied by the above construction since we do not interfere with the generation of the OTA note.

Note and Ciphertext Privacy. Privacy mandates note and ciphertext hiding as well as note and encryption anonymity. If the underlying OTA scheme satisfies this, then the above construction trivially achieves it too. This is due to the fact that we do not interfere with note generation and that all the additional values are hidden by encrypting them using the underlying OTA encryption procedure.

Note Uniqueness. Note uniqueness captures that honestly generated notes (aka addresses) do not collide, except with negligible probability. This is obviously fulfilled by our construction.

Nullifier Uniqueness and collision resistance. Nullifier uniqueness demands that for the same note, no two nullifiers can be constructed and that the probability of two nullifiers colliding is negligible. As above, this is retained by the construction if the underlying OTA scheme satisfies it.

Nullifier security. The most crucial change of our construction is the nullifier. We gain efficiency by including signatures and (re-randomized) keys as part of the nullifier, but we trade the strong pseudo-random property, which has some security implications (compared to the strawman approach).

If the creator of a note is honest, the corresponding owner is able to spend the note in a private and anonymous way. In particular, if the PCS recipient key is re-randomized accordingly, no linking within the transaction log is possible thanks to the hiding and unlinkability property of PCS and the security of the underlying OTA scheme. The transaction log only reveals that parties are transacting which are allowed to transact by the policy. On the other hand, if different notes are created for the same PCS receiver key, then the only information that leaks from this, is the fact that the same party must, again, be transacting—but no link exists to the actual note or other transactions that use a re-randomized receiver key of this party, thanks to the privacy of the commitment scheme, the unlinkability of the sender PCS key (which by default gets re-randomized), and the security of the underlying OTA scheme.

However, if the creator of a note is malicious, then this creator (and only this creator) has enough information to determine that the owner of the note has been spending the note in a transaction. This is due to the presence of the additional PCS-related values that are revealed (as part of the nullifier). It is however possible to remedy this situation proactively, namely by spending the note to itself using a freshly randomized PCS key as soon as the transaction appears in the log. This is incidentally one of the recommended measures by Zcash to achieve *everlasting anonymity* [11].

Finally, we observe that spending a note is only possible if a party has access to the OTA private key and the PCS private key (which follows from the security provided by the underlying OTA nullifier and the unforgeability of the PCS signature on this nullifier).

7.3 Distributed Setup and User Enrollment

In credential systems, issuance is often distributed across a set of servers to avoid a single point of failure. Such failures may for example include the leakage or malicious revelation of the master secret key. Hence, the security of the system is improved if the system's setup values and user enrollment are implemented by distributed processes with the property that only a large collusion of servers would be able to recreate crucial secret values. This is important in our context, because the revelation of the master secret key would (necessarily) limit the achievable level of attribute hiding in practice, as it allows an attacker to self-issue credentials and determine w.r.t. which participants it can generates valid signatures. This results in potentially arbitrary loss in privacy.

In Appendix E, we showcase how our constructions can be implemented in the distributed setting. In general, the idea is to have the master secret-key shared among the servers (plus additional shared randomness), and have a client obtain partial results $r_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}_i, x)$, and perform client-side aggregation to reconstruct the full output of KeyGen. Such a process ensures that, unless a certain threshold of servers collude (e.g. up to $n-1$ in an honest-but-curious scenario), the CA's have no advantage over any other party in the system.

7.4 Application to Larger Systems and CBDCs

Zcash is a decentralized anonymous payment system and, while the considerations above capture the technical aspects of how to integrate PCS with private transaction systems, it is important to note that PCS can significantly improve the privacy for users in more complex, compliance-seeking systems, be it centralized or decentralized. Central-bank digital currencies have received a lot of attention in recent years. A core requirement [50] in these systems is the so-called *comprehensive regulatory compliance* which puts restrictions and requirements on (1) the number of coins in circulation, (2) sending and receiving limits, (3) transaction value limits, (4) privacy, (5) accountability and (6) auditability.

A critical feature is accountability and auditability [24]—how can a regulator be assured everything is complying with the jurisdiction? A somewhat standard technique to achieve auditability is by privacy revocation techniques [50]: in case of suspicious activities, a user can be traced and its privacy can be revoked completely. Whether a user is considered suspicious is decided outside of the technical system, whereas the technical system enables, for example, an auditor to view keys and unmask any transaction of any user. For improved resilience, the revocation capability is supposed to be shared by a group of anonymity revokers such that a quorum is needed to unlock the feature. It goes without saying that, while this approach trivially ensures that an auditor can learn anything it needs to know to perform its task, this immense power comes with a huge risk for users. Not only is there the danger of false accusation and the revelation of an individual's activities even if they were legal, but this also opens the door for pro-active surveillance. This puts the users into a weak position and undermines the right for privacy in a rather extreme way.

Acknowledging that the other extreme, unconditional anonymity, is problematic as well, the research on *accountable privacy* has been picking up steam that tries to balance privacy, accountability, and auditability, such that users enjoy much stronger guarantees. Considering the UTT system as an example of such an approach in the DAP domain [59], that implements so-called budget coins which can be spent (unconditionally) private, yet accountable. This is due to the fact that the budget coin is governed by a spending limit, thus representing a digital analogue of cash. Auditiability is achieved by having a user fill up its budget in regular intervals by presenting credentials to the auditor. All remaining transaction are potentially subject to privacy revocation. As described in Section 1, the general approach is to define several types of assets, a strategy taken in [33], or different types of transactions for which different rules apply. In the CBDC domain, Platypus [64] also proposes such a path. Therefore, if there is a cryptographic way to ensure that a certain policy for a given asset is fulfilled, there is no need for revocation, and transactions can be made untraceable without harming anyone. For all other cases, traceable/revocable transactions are still a good fit. This design gives a user much stronger privacy guarantees for all policies that can be enforced on a technical level.

The compliance requirements (1)-(3) have received a lot of attention for automatic enforcement, since they affect the transaction content and are thus comparably simpler to handle. However, a lot of the need for identity revocation stems form the lack of cryptographic policy enforcement that involves relevant properties of sender and receiver, including age, citizenship, place of residency, more technical attributes like governing (tax) jurisdiction or financial score, and more generally certified attributes by external auditors (cf. the chosen policy classes for separable policies or RBAC, and of course the richer set computable by inner-product predicates). A PCS public key is the digital, private-preserving representation of a user's relevant credentials while signatures between two users prove compliance. A user is furthermore free to change its representation to ensure unlinkability. We obtain the following: coupling ul-PCS with any system such as UTT or Platypus allows for a transaction system that (1) limits the number of coins (2) enforces sending and receiving limits and transaction value limits, and (3) achieves strong accountable privacy for a rich class of policies without the need for revocation. The capability to revoke is thus pushed to the boundary, i.e., to the edge cases which are not clearly governed by a reasonable (digital) policy.

Due to its low-level nature of being a signature scheme tied to digital credentials, there is a lot of flexibility in the usage of a PCS scheme. For example, if an application requires traceability or revocation, the user is still free to follow standard procedures to register its public key with a PKI to bind it to a real-world identity, or to secret share its private key with revocation servers that would enable traceability. Having a PKI in place can assist in disincentivizing users from sharing private keys, if that is deemed a concern, as well as any standard technique, such as PKI-assured non-transferability, can be used for this purpose [18] just like with ordinary credential or signature systems.

Acknowledgments. The second author is supported in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by CyberSecurity Research Flanders with reference number VR20192203. Part of this work was done while the second author was an intern at Mysten Labs. The third author is supported by an MC2 Postdoctoral Fellowship.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7_12
2. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. Journal of Cryptographic Engineering **3**(2), 111–128 (Jun 2013). https://doi.org/10.1007/s13389-013-0057-3
3. Alonso, K.M., Joancomartí, J.H.: Monero - privacy in the blockchain. Cryptology ePrint Archive, Report 2018/535 (2018), https://eprint.iacr.org/2018/535
4. Androulaki, E., Camenisch, J., Caro, A.D., Dubovitskaya, M., Elkhiyaoui, K., Tackmann, B.: Privacy-preserving auditable token payments in a permissioned blockchain system. In: AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020. pp. 255–267. ACM (2020). https://doi.org/10.1145/3419614.3423259
5. Ateniese, G., Francati, D., Nuñez, D., Venturi, D.: Match me if you can: Matchmaking encryption and its applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 701–731. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26951-7_24
6. Badertscher, C., Matt, C., Waldner, H.: Policy-compliant signatures. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part III. LNCS, vol. 13044, pp. 350–381. Springer, Heidelberg (Nov 2021). https://doi.org/10.1007/978-3-030-90456-2_12
7. Badertscher, C., Maurer, U., Portmann, C., Rito, G.: Revisiting (R)CCA security and replay protection. In: Garay, J. (ed.) PKC 2021, Part II. LNCS, vol. 12711, pp. 173–202. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75248-4_7
8. Barki, A., Gouget, A.: Achieving privacy and accountability in traceable digital currency. Cryptology ePrint Archive, Paper 2020/1565 (2020), https://eprint.iacr.org/2020/1565, https://eprint.iacr.org/2020/1565
9. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11693383_22
10. Ben-Or, M., Goldreich, O., Goldwasser, S., Håstad, J., Kilian, J., Micali, S., Rogaway, P.: Everything provable is provable in zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 37–56. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34799-2_4
11. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.36
12. Blom, F., Bouman, N.J., Schoenmakers, B., de Vreede, N.: Efficient secure ridge regression from randomized gaussian elimination. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5. pp. 301–316. Springer (2021)
13. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_4

14. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_13
15. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_30
16. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (Feb 2007). https://doi.org/10.1007/978-3-540-70936-7_29
17. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
18. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_7
19. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28628-8_4
20. Cankaya, E.C.: Bell-LaPadula Confidentiality Model, pp. 71–74. Springer US, Boston, MA (2011). https://doi.org/10.1007/978-1-4419-5906-5_773, https://doi.org/10.1007/978-1-4419-5906-5_773
21. Cecchetti, E., Zhang, F., Ji, Y., Kosba, A.E., Juels, A., Shi, E.: Solidus: Confidential distributed ledger transactions via PVORM. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 701–717. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134010
22. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 281–300. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_18
23. Chatzigiannis, P., Baldimtsi, F.: MiniLedger: Compact-sized anonymous and auditable distributed payments. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021, Part I. LNCS, vol. 12972, pp. 407–429. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-88418-5_20
24. Chatzigiannis, P., Baldimtsi, F., Chalkias, K.: SoK: Auditability and accountability in distributed payment systems. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 21, Part II. LNCS, vol. 12727, pp. 311–337. Springer, Heidelberg (Jun 2021). https://doi.org/10.1007/978-3-030-78375-4_13
25. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (oct 1985). https://doi.org/10.1145/4372.4373, https://doi.org/10.1145/4372.4373
26. Chen, Y., Ma, X., Tang, C., Au, M.H.: Pgc: Decentralized confidential payment system with auditability. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) Computer Security – ESORICS 2020. pp. 591–610. Springer International Publishing, Cham (2020)
27. Crites, E.C., Kohlweiss, M., Preneel, B., Sedaghat, M., Slamanig, D.: Threshold Structure-Preserving Signatures. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14439, pp. 348–382. Springer (2023). https://doi.org/10.1007/978-981-99-8724-5_11, https://doi.org/10.1007/978-981-99-8724-5_11
28. Damgård, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., Siniscalchi, L.: Balancing privacy and accountability in blockchain identity management. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 552–576. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_23
29. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (Jan 2005). https://doi.org/10.1007/978-3-540-30580-4_28
30. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (Aug 1984)
31. Engelmann, F., Kerber, T., Kohlweiss, M., Volkhov, M.: Zswap: zk-snark based non-interactive multi-asset swaps. Proc. Priv. Enhancing Technol. **2022**(4), 507–527 (2022). https://doi.org/10.56553/popets-2022-0120, https://doi.org/10.56553/popets-2022-0120
32. Escala, A., Groth, J.: Fine-tuning Groth-Sahai proofs. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 630–649. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0_36
33. EspressoSystems: Specification: Configurable asset privacy. GitHub, https://github.com/EspressoSystems/cap/blob/main/cap-specification.pdf (2022), https://github.com/EspressoSystems/cap/blob/main/cap-specification.pdf
34. FATF: International standards on combating money laundering and the financing of terrorism and proliferation. https://www.fatf-gafi.org/en/publications/Fatfrecommendations/Fatf-recommendations.html (2023)

35. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
36. Fortnow, L.: The complexity of perfect zero-knowledge (extended abstract). In: Aho, A. (ed.) 19th ACM STOC. pp. 204–209. ACM Press (May 1987). https://doi.org/10.1145/28395.28418
37. Fuchsbauer, G., Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. Journal of Cryptology **32**(2), 498–546 (Apr 2019). https://doi.org/10.1007/s00145-018-9281-4
38. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: A cryptographic investigation of Mimblewimble. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 657–689. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_22
39. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008). https://doi.org/https://doi.org/10.1016/j.dam.2007.12.010, applications of Algebra to Cryptography
40. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer, Heidelberg (Feb 2016)
41. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986)
42. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). https://doi.org/10.1145/28395.28420
43. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (Apr 1988)
44. Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 192–208. Springer, Heidelberg (Aug 2009). https://doi.org/10.1007/978-3-642-03356-8_12
45. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_24
46. Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 491–511. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45611-8_26
47. Herold, G., Hoffmann, M., Klooß, M., Ràfols, C., Rupp, A.: New techniques for structural batch verification in bilinear groups with applications to groth-sahai proofs. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1547–1564. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134068
48. Karantaidou, I., Baldimtsi, F.: Efficient constructions of pairing based accumulators. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–16. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00033
49. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_9
50. Kiayias, A., Kohlweiss, M., Sarencheh, A.: Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 1739–1752. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560707, https://doi.org/10.1145/3548606.3560707
51. Maurer, U.M.: Unifying zero-knowledge proofs of knowledge. In: Preneel, B. (ed.) AFRICACRYPT 09. LNCS, vol. 5580, pp. 272–286. Springer, Heidelberg (Jun 2009)
52. Maxwell, G.: CoinJoin: Bitcoin privacy for the real world. https://bitcointalk.org/?topic=279249 (2013)
53. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009), whitepaper, http://bitcoin.org/bitcoin.pdf
54. Narula, N., Vasquez, W., Virza, M.: Zkledger: Privacy-preserving auditing for distributed ledgers. In: Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation. p. 65–80. NSDI'18, USENIX Association, USA (2018)
55. Okamoto, T., Takashima, K.: Adaptively attribute-hiding (hierarchical) inner product encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 591–608. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_35

56. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
57. Rogaway, P., Zhang, Y.: Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 3–32. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_1
58. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_22
59. Tomescu, A., Bhat, A., Applebaum, B., Abraham, I., Gueta, G., Pinkas, B., Yanai, A.: UTT: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Report 2022/452 (2022), https://eprint.iacr.org/2022/452
60. Tran, M., Luu, L., Kang, M.S., Bentov, I., Saxena, P.: Obscuro: A bitcoin mixer using trusted execution environments. In: Proceedings of the 34th Annual Computer Security Applications Conference. p. 692–701. ACSAC '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3274694.3274750, https://doi.org/10.1145/3274694.3274750
61. Unlinkable_PCS: The implementation of our construction. https://github.com/Mahdi171/Unlinkable_PCS
62. W3C: Verifiable credentials data model v1.1. https://www.w3.org/TR/vc-data-model/ (2022), https://www.w3.org/TR/vc-data-model/
63. Wüst, K., Kostiainen, K., Capkun, V., Capkun, S.: PRCash: Fast, private and regulated transactions for digital currencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 158–178. Springer, Heidelberg (Feb 2019). https://doi.org/10.1007/978-3-030-32101-7_11
64. Wüst, K., Kostiainen, K., Delius, N., Capkun, S.: Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy-Preserving Regulation. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 2947–2960. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560617, https://doi.org/10.1145/3548606.3560617

A Note on Multi-Challenge Unlinkability

The definition of multi-challenge unlinkability is almost the same as the definition of unlinkability with the only difference that, instead of submitting a single challenge query x, the adversary has access to a key generation oracle $\mathsf{QKeyGen}$ that it can query using multiple attributes to obtain multiple challenge public keys. Additionally, the adversary can query the rerandomization oracle using an index i to obtain a rerandomized key for the public key associated with the index i. If the adversary wants to obtain the corresponding secret key for a public key, it can query the corruption oracle QCor using the corresponding index i. The signing oracle in this case QSign takes the same inputs as in the unforgeability and the attribute-hiding game. More formally:

Definition 21. *Let* $\mathsf{ULPCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a ul-PCS scheme that satisfies the detectability property. For* $\beta \in \{0, 1\}$*, we define the experiment* $\mathsf{MC\text{-}Link}_\beta^{\mathsf{ULPCS}}$ *in Figure 21. The advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is defined by*

$$\mathsf{Adv}_{\mathsf{ULPCS},\mathcal{A}}^{\mathsf{MC\text{-}Link}}(\lambda) = |\Pr[\mathsf{MC\text{-}Link}_0^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{MC\text{-}Link}_1^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A}) = 1]|.$$

An adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *is called valid, if no index* i *is queried to* both *oracles* $\mathsf{QRandKey}_\beta(i)$ *and* $\mathsf{QCor}(i)$*.*

We call such a ul-PCS scheme ULPCS unlinkable *if for any polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}_{\mathsf{ULPCS},\mathcal{A}}^{\mathsf{MC\text{-}Link}}(\lambda) \leq \mathsf{negl}(\lambda)$*.*

$\mathsf{MC\text{-}Link}^{\mathsf{ULPCS}}_{\beta}(1^\lambda, \mathcal{A})$
$(F, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, F)$
$\alpha \leftarrow \mathcal{A}_2^{\mathsf{QKeyGen}(\cdot),\mathsf{QRandKey}_\beta(\cdot),\mathsf{QCor}(\cdot),\mathsf{QSign}(\cdot,\cdot,\cdot)}(\mathsf{mpk}, \mathsf{st}_1)$
Output: α

Fig. 21: Many-Challenges Unlinkability game of ULPCS.

It is straightforward to verify that the single-challenge implies the multi-challenge extension. Formally, this extension is formalized by introducing the oracles QKeyGenC and QSign_β and defining the multi-challenge version in Figure 21. In the game, we maintain an additional set $\mathcal{QCK}$ (initially empty):

Theorem 4 (Link implies MC-Link). *Let* ULPCS *be* Link *secure, then* ULPCS *is also* MC-Link *secure.*

Proof (Sketch). This proof proceeds using a simple hybrid argument using the following game:

Game G_k: For the first k keys that are being queried to the rerandomization oracle $\mathsf{QRandKey}_\beta$, fresh keys are generated, whereas for the remaining keys all queries asked to $\mathsf{QRandKey}_\beta$ are answered using rerandomized keys.

Let Q be the number of overall key queries, then it holds that

$$\mathsf{MC\text{-}Link}_0 = G_0 \approx \cdots \approx G_Q = \mathsf{MC\text{-}Link}_1$$

To conclude the proof, it needs to be shown that $G_{k-1} \approx G_k$ for all $k \in [Q]$. This can be done using a reduction to the Link security game by forwarding the k'th challenge query to the underlying challenger of the Link game and then reply using the obtained key. The remaining keys are generated using key generation queries to the underlying challenger. The obtained secret keys can then be used to answer potential corruption queries of the adversary. To answer signing queries, they are also directly forwarded to the underlying challenger or generated using the known secret keys.

Therefore it follows that $G_{k-1} \approx G_k$ for all $k \in [Q]$, which proves the theorem. □

B Security Analysis

Here, we present the formal proof of the ul-PCS scheme for generic policies (Theorem 1). We prove three theorems in this supplement where each theorem covers one aspect, i.e., unforgeability, attribute-hiding, and unlinkability, respectively. Furthermore, we also argue the detactability of the schemes. For the sake of notation, we denote the unlinkable PCS scheme for generic policies $F(x, y)$ by ULPCS. The concrete specification as pseudo-code can be found in the submission.

The proofs of Theorems 2 and 3 for separable and role-based policies, respectively, are given by describing which arguments need to be adjusted to accommodate the replacement of the PE scheme in these constructions. We describe these adjustments for unforgeability, attribute-hiding, and unlinkability right after the proofs of the generic scheme.

B.1 Detectability

The detect algorithm Detect behaves the same in all of the three different schemes. It takes as an input the master public key mpk, the challenge public key pk^* as well as the lists $(Q_1, \ldots, Q_c)$. It then behaves as follows: for all $i \in [c]$, it generates the maximal amount of rerandomizations. In more detail, for all $i \in [c]$, it executes as many rerandomizations of the keys contained in $Q_i[]$ until $Q_i[]$ contains T_{Rand} keys. Afterwards, it searches all the lists $Q_i[]$ and if it finds an index pair (i, i') for which it holds that $Q_i[i'] = (\mathsf{pk}^*, \mathsf{sk}^*)$, then it adds i to its final list Q. After Det has iterated over all lists $(Q_1, \ldots, Q_c)$, we distinguish between three cases: first, Q only contains a single i, second, Q contains multiple i's and, third, Q is empty. In the first case, Det simply outputs the single i, in the second case, Det outputs the lower of the two indices contained in Q and, in the third case, Det outputs $\perp$. To argue the correctness of Det, we need to analyze the three different cases. We start by analyzing the third case. The third case can never occur because the key pk^* is generated by checking $Q_i[j]$ and therefore the detect algorithm Det will also find this index pair. In the first case, Det behaves correct since there is only a single index pair which explains the key pk^* and this is output by Det. The second case, can only occur if a key collision has happened as defined in the event $\mathsf{KeyColl}_{\mathcal{A}}$ below, which is negligible due to the security of the PRF (see below for the argument). Therefore, it follows that the algorithm Det is correct with probability $1 - \mathsf{negl}(\lambda)$, which concludes the detectability argument.

B.2 Unforgeability

Theorem 5. *Let* $T_{\mathsf{Rand}} = \mathsf{poly}(\lambda)$*. If* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *is an* $\mathsf{EUF\text{-}CMA}$*-secure signature scheme,* PRF *a secure pseudorandom function,* $\mathsf{NIZK}_{\mathcal{L}_1} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *a knowledge sound proof system for language* $\mathcal{L}_1$ *and* $\mathsf{NIZK}_{\mathcal{L}_2} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is a knowledge sound proof system for language* $\mathcal{L}_2$*, then* ULPCS *described in Figures 14 and 14 is* T_{Rand} $\mathsf{EUF\text{-}CMA}$ *secure, i.e. it holds that* $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{ULPCS},\mathcal{A}} = \mathsf{negl}(\lambda)$*.*

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

- Event $\mathsf{KeyColl}_{\mathcal{A}}$: The adversary $\mathcal{A}$ terminates and it holds that there are indices i, i', j, j' with $i \neq j$ or $i' \neq j'$ such that $((i, i'), \mathsf{pk}_i, \cdot, \cdot), ((j, j'), \mathsf{pk}_j, \cdot, \cdot) \in \mathcal{QK}$, where $\mathsf{pk}_i = (\mathsf{ID}_i, \ldots)$, $\mathsf{pk}_j = (\mathsf{ID}_j, \ldots)$, for which $\mathsf{ID}_i = \mathsf{ID}_j$.
- Event $\mathsf{KeyForge}_{\mathcal{A}}$: The adversary $\mathcal{A}$ terminates with output $(\mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)$ and there exists an entry $(\cdot, \mathsf{pk}^*_S, \mathsf{pk}^*_R, m^*, \sigma^*) \in \mathcal{QS} \cup \{(\mathsf{pk}_S, \mathsf{pk}_R, m, \sigma)\}$ for which the following

condition holds: $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}_S^*, \mathsf{pk}_R^*, m^*, \sigma^*) = 1 \wedge (S = \bot \vee R = \bot)$ where $S \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}_S^*, (\mathcal{QK}_1, \dots, \mathcal{QK}_{i_{\max}}))$ and $R \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}_R^*, (\mathcal{QK}_1, \dots, \mathcal{QK}_{i_{\max}}))$.

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

– Event $\mathsf{WIN1}_{\mathcal{A}}$: The adversary generates the output $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge \exists (i,j), \mathsf{sk}, x\ \forall (i', j'), \sigma :\ ((i,j), \mathsf{pk}, \mathsf{sk}, x) \in \mathcal{QK} \setminus \mathcal{QC} \wedge ((i',j'), \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma) \notin \mathcal{QS}$.
– Event $\mathsf{WIN2}_{\mathcal{A}}$: The adversary $\mathcal{A}$ generates the output $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ for which it holds that $\mathsf{Verify}(\mathsf{mpk}, \mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*) = 1 \wedge \big[(S \neq \bot) \wedge (R \neq \bot) \Rightarrow F(x_S, x_R) = 0\big]$ where $S \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}, (\mathcal{QK}_1, \dots, \mathcal{QK}_{i_{\max}}))$, $R \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}^*, (\mathcal{QK}_1, \dots, \mathcal{QK}_{i_{\max}}))$ and x_S and x_R denote the respective attributes.

By Lemma 2 and Lemma 3, we obtain

$$\Pr[\mathsf{KeyForge}_{\mathcal{A}}] = \mathsf{negl}(\lambda) \text{ and } \Pr[\mathsf{KeyColl}_{\mathcal{A}}] = \mathsf{negl}(\lambda)$$

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

Finally, we obtain by Lemma 4 and by Lemma 5 that

$$\begin{aligned} &\Pr[\mathsf{WIN1}_{\mathcal{A}}] = \mathsf{negl}(\lambda) \text{ and} \\ &\Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}] = \mathsf{negl}(\lambda). \end{aligned}$$

By definition of the events, we have

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

This concludes the proof of the theorem. □

Lemma 2. *It holds that* $\Pr[\mathsf{KeyColl}_{\mathcal{A}}] = \mathsf{negl}(\lambda)$.

Proof. To bound the probability for the occurrence of $\mathsf{KeyColl}_{\mathcal{A}}$, we need to bound the probability that there exist two honestly generated/rerandomized keys $\mathsf{pk} := (\mathsf{ID}, \dots)$ and $\mathsf{pk}' := (\mathsf{ID}', \dots)$ with $\mathsf{ID} = \mathsf{ID}'$. The ID of an honestly generated key is generated using a PRF evaluation as well as an attached zero-knowledge proof that proves that the resulting string is indeed an honest PRF evaluation. By relying on the soundness of the zero-knowledge proof, it is ensured that the resulting ID is indeed a valid PRF evaluation, which, by the n-instance/parallel composable security of the PRF, allows us to consider the ID's in this analysis as randomly sampled. Therefore, to conclude the proof of the lemma, it suffices to bound the collision probability for randomly sampled identities.

In our setting, we have n different keys that are being generated, where each of those keys can be randomized T times. This means that overall $n{\cdot}T$ different ID's are being sampled. The probability that all of these ID's are different is $(1-\frac{1}{2^\lambda})\cdot(1-\frac{2}{2^\lambda})\cdot\dots\cdot(1-\frac{n\cdot T-1}{2^\lambda})\prod_{k=1}^{nT-1}(1-\frac{k}{2^\lambda})$. For this probability it holds that $\prod_{k=1}^{nT-1}(1-\frac{k}{2^\lambda}) \geq (1-\frac{nT-1}{2^\lambda})^{nT-1}$, which, in turn, can be bounded using Bernoulli's inequality $(1-\frac{nT-1}{2^\lambda})^{nT-1} \geq 1-(nT-1)\cdot\frac{nT-1}{2^\lambda} = 1-\frac{(nT-1)^2}{2^\lambda}$. Considering now the complementary event that at least one collision of ID's occurs, then the resulting probability for this event is equal to $1-(1-\frac{(nT-1)^2}{2^\lambda}) = \frac{(nT-1)^2}{2^\lambda}$, which is negligible in λ. This concludes the proof of the lemma. $\square$

Lemma 3. *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *be an* $\mathsf{EUF\text{-}CMA}$*-secure signature scheme and* $\mathsf{NIZK}_{\mathcal{L}_1} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is a knowledge sound proof system for* $\mathcal{L}_1$*, then* $\Pr[\mathsf{KeyForge}_{\mathcal{A}}] = \mathsf{negl}(\lambda)$.

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

We first make a first transition to a hybrid world $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}_{\mathrm{Hyb}}$, which is identical to $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}$ except that we replace $\mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Setup}(1^\lambda)$ by the CRS simulation algorithm Ext_1 associated to the NIZK scheme which also outputs the state $\mathsf{st}_{\mathsf{Rand}}$ for the second extraction algorithm Ext_2. All above defined events are still defined in this hybrid experiment. It follows directly from the knowledge soundness property of the NIZK, using a standard reduction, that

$$\Pr[\mathsf{KeyForge}_{\mathcal{A}}] \leq \Pr_{\mathrm{Hyb}}\left[\,\mathsf{KeyForge}_{\mathcal{A}}\,\right] + \mathsf{negl}(\lambda),$$

where $\Pr_{\mathrm{Hyb}}[.]$ makes explicit that this probability is taken w.r.t. experiment $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}_{\mathrm{Hyb}}$.

Now, to bound the probability of the occurrence of $\mathsf{KeyForge}$, we need to bound three different subcases:

1. The adversary is not able to forge a signature σ^1_{sig} or σ^2_{sig} that would suffice as a proof for the relation $R_{\mathcal{L}_1}$.
2. The adversary is not able to forge a signature σ_{k+1} that would suffice as a proof for the relation $R_{\mathcal{L}_1}$.
3. The adversary is not able to break the soundness of the underlying $\mathsf{NIZK}_{\mathcal{L}_1}$ to generate a valid proof without being in possession of a witness.

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

The adversary $\mathcal{B}$ behaves using the algorithms described in the protocol with the only difference that it does not generate the key pair $(\mathsf{vk}^A_{\mathsf{sig}}, \mathsf{sk}^A_{\mathsf{sig}})$ on its own but obtains it from an underlying challenger. Also the corresponding signatures $\sigma^1_{\mathsf{sig}}, \sigma^2_{\mathsf{sig}}$ and σ^3_{sig} , that are the outputs of key generation queries, are not generated by $\mathcal{B}$ directly but through signing oracle queries of $\mathcal{B}$ to its underlying challenger.

When $\mathcal{A}$ terminates with $(\mathsf{pk}_S^* := (\mathsf{ID}_S^*, \mathsf{vk}_S^*, \mathsf{ct}_S^*, \pi_S^*), \mathsf{pk}_R^* := (\mathsf{ID}_R^*, \mathsf{vk}_R^*, \mathsf{ct}_R^*, \pi_R^*), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}_1$ first checks whether the conditions of event $\mathsf{KeyForge}_{\mathcal{A}}$ holds, using the detect procedure which will output S' and R'. If the conditions of $\mathsf{KeyForge}_{\mathcal{A}}$ do not hold, then abort. For the remainder of the proof we assume that, WLOG the condition is fulfilled w.r.t. S'. The R' case follows accordingly.

If the conditions of event $\mathsf{KeyForge}_{\mathcal{A}}$ are fulfilled, then $\mathcal{B}$ calls $(\mathsf{usk}^*, \sigma^*) \leftarrow \mathsf{Ext}_2(\mathsf{CRS}_{\mathsf{Rand}}, \mathsf{st}_{\mathsf{Rand}}, (T_{\mathsf{Rand}}, \mathsf{ID}_S^*, \mathsf{vk}_S^*, \mathsf{ct}_S^*, \mathsf{vk}_{\mathsf{sig}}^{A}, \mathsf{mpk}_{\mathsf{PE}}), \pi_S^*))$ and checks whether $(x := (T_{\mathsf{Rand}}, \mathsf{ID}_S^*, \mathsf{vk}_S^*, \mathsf{ct}_S^*, \mathsf{vk}_{\mathsf{sig}}^{A}, \mathsf{mpk}_{\mathsf{PE}}), w := (\mathsf{usk}^*, \sigma^*)) \in R_{\mathcal{L}_1}$ (which is efficiently checkable). Afterwards, $\mathcal{B}$ parses $\mathsf{usk}^* := (\mathsf{k}^*, \mathsf{vk}_{\mathsf{sig}}^*, \mathsf{sk}_{\mathsf{sig}}^*, \sigma_{\mathsf{sig}}^{*,1}, \sigma_{\mathsf{sig}}^{*,2}, \sigma_{\mathsf{sig}}^{*,3}, x^*, \mathsf{sk}_{f_x}^*)$ it checks if $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A}, (\mathsf{k}^*, x^*), \sigma_{\mathsf{sig}}^{*,1}) = 1$ or $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A}, (\mathsf{k}^*, \mathsf{vk}_{\mathsf{sig}}^*), \sigma_{\mathsf{sig}}^{*,2}) = 1$ and submits the corresponding message-signature-pair that verifies, i.e. either $((\mathsf{k}^*, x^*), \sigma_{\mathsf{sig}}^{*,1})$ or $((\mathsf{k}^*, \mathsf{vk}_{\mathsf{sig}}^*), \sigma_{\mathsf{sig}}^{*,2})$, to its challenger if it has not been previously output by the signing oracle. Otherwise, it aborts.

Before we analyze what happens in the case that $(x, w) \notin R_{\mathcal{L}_1}$, we need to bound the case where the adversary $\mathcal{A}$ outputs a forgery for the signature σ_{k+1}. This part of the proof, i.e. the adversary $\mathcal{B}$ in this case, almost behaves as before, with the only difference that the adversary $\mathcal{B}$ randomly samples a value $i \leftarrow [q]$, where q is the number of key generation queries asked by the adversary $\mathcal{A}$, receives $\mathsf{vk}_{\mathsf{sig}}$ from the underlying challenger and uses $\mathsf{vk}_{\mathsf{sig}}$ from the challenger to answer the i'th key generation query asked by $\mathcal{A}$. To finish the key generation and for further rerandomization queries that are asked for the i'th key, the adversary $\mathcal{B}$ uses the signing oracle of its underlying challenger. When $\mathcal{A}$ terminates with $(\mathsf{pk}_S^* := (\mathsf{ID}_S^*, \mathsf{vk}_S^*, \mathsf{ct}_S^*, \pi_S^*), \mathsf{pk}_R^* := (\mathsf{ID}_R^*, \mathsf{vk}_R^*, \mathsf{ct}_R^*, \pi_R^*), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}_1$ first checks whether the conditions of event $\mathsf{KeyForge}_{\mathcal{A}}$ holds, using the detect procedure which will output S' and R'. If the conditions of $\mathsf{KeyForge}_{\mathcal{A}}$ do not hold, then it aborts. Also, as described above, we assume that, WLOG the condition is fulfilled w.r.t. S'. The R' case follows accordingly. If the conditions of event $\mathsf{KeyForge}_{\mathcal{A}}$ are fulfilled, then $\mathcal{B}$ calls $(\mathsf{usk}^*, \sigma^*) \leftarrow \mathsf{Ext}_2(\mathsf{CRS}_{\mathsf{Rand}}, \mathsf{st}_{\mathsf{Rand}}, (T_{\mathsf{Rand}}, \mathsf{ID}_S^*, \mathsf{vk}_S^*, \mathsf{ct}_S^*, \mathsf{vk}_{\mathsf{sig}}^{A}, \mathsf{mpk}_{\mathsf{PE}}), \pi_S^*)$, checks whether $(x := (T_{\mathsf{Rand}}, \mathsf{ID}_S^*, \mathsf{vk}_S^*, \mathsf{ct}_S^*, \mathsf{vk}_{\mathsf{sig}}^{A}, \mathsf{mpk}_{\mathsf{PE}}), w := (\mathsf{usk}^*, \sigma^*)) \in R_{\mathcal{L}_1}$ (which is efficiently checkable) and if S' identified by Detect corresponds to the key that has been generated as the answer to the i'th query. Afterwards, $\mathcal{B}$ checks if $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}, (\mathsf{pk}_S^* \| \mathsf{ID}_S^*), \sigma^*) = 1$ and submits the signature σ^*, if it passes the test and has not been previously output by the signing oracle of the underlying challenger, as a forgery. Otherwise, it aborts. To conclude the analysis we argue that the above described case occurs with probability $\frac{1}{q}$, which is exactly the probability that the adversary $\mathcal{B}$ guesses the index for the rerandomized key correctly.

If $(x, w) \notin R_{\mathcal{L}_1}$ then abort with failure event $\mathsf{Fail}_{\mathrm{ext}}$. Therefore, taking into account the two reductions described above, it holds that the advantage can be reduced to the unforgeability of the underlying signature scheme with probability $\Pr_{\mathrm{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\mathrm{ext}}}\right]$. This, in turn, results in the fact that $\Pr_{\mathrm{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}}\right] = \Pr_{\mathrm{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\mathrm{ext}}}\right] + \Pr_{\mathrm{Hyb}}\left[\mathsf{Fail}_{\mathrm{ext}}\right] + \mathsf{negl}(\lambda)$.

Since a forgery for the underlying $\mathsf{EUF\text{-}CMA}^{\mathsf{DS}}$ experiment only occurs with negligible probability, it follows that $\Pr_{\mathrm{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\mathrm{ext}}}\right] = \mathsf{negl}(\lambda) + \frac{1}{q}\mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$ (after the two analysis above) and, to conclude the proof, it only remains to show that

$\Pr_{\mathrm{Hyb}}[\,\mathsf{Fail}_{\mathrm{ext}}\,] = \mathsf{negl}(\lambda)$. This can be done by relying on the soundness property of the underlying $\mathsf{NIZK}_{\mathcal{L}_1}$ as mentioned in the second of the two cases above.

To conclude the proof, it remains to show that $\Pr[\mathsf{Fail}_{\mathsf{Ext}}] = \mathsf{negl}(\lambda)$. Also here, we assume that, WLOG the condition is fulfilled w.r.t. S'. The R' case follows accordingly. Our adversary $\mathcal{B}'$ for this case receives as an input the $\mathsf{CRS}_{\mathsf{Rand}}$ and executes the same instructions as $\mathcal{B}$, with the exceptions that it generates $(\mathsf{vk}^{A}_{\mathsf{sig}}, \mathsf{sk}^{A}_{\mathsf{sig}})$ by itself and uses it to generate the corresponding signatures by itself. Additionally, when $\mathcal{A}$ terminates with output $(\mathsf{pk}^*_S := (\mathsf{ID}^*_S, \mathsf{vk}^*_S, \mathsf{ct}^*_S, \pi^*_S), \mathsf{pk}^*_R := (\mathsf{ID}^*_R, \mathsf{vk}^*_R, \mathsf{ct}^*_R, \pi^*_R), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}'$ behaves as $\mathcal{B}$ without running the extractor. Instead, it just outputs $(x := (T_{\mathsf{Rand}}, \mathsf{ID}^*_S, \mathsf{vk}^*_S, \mathsf{ct}^*_S, \mathsf{vk}^{A}_{\mathsf{sig}}, \mathsf{mpk}_{\mathsf{PE}}), \pi^*_S)$ in case the conditions of $\mathsf{KeyForge}_{\mathcal{A}}$ are satisfied (note that the extractor is run as part of the knowledge soundness experiment). As above, the emulation towards $\mathcal{A}$ is perfect until the point where $\mathcal{B}'$ would abort. This results in the claimed advantage since the event of interest is that the extractor Ext_2 is called precisely on the accepting proof string π^*_S output by $\mathcal{A}$ which produces a witness w but for which $(x, w) \notin R_{\mathcal{L}_1}$. This concludes the proof of the lemma. $\square$

Lemma 4. *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *be an* $\mathsf{EUF\text{-}CMA}$*-secure signature scheme, then* $\Pr[\mathsf{WIN1}_{\mathcal{A}}] = \mathsf{negl}(\lambda)$.

Proof. To prove this lemma, we construct an adversary $\mathcal{B}$ that simulates $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}$ towards $\mathcal{A}$. We show that if $\mathcal{A}$ outputs $(\mathsf{pk}, \mathsf{pk}^*, m^*, \sigma^*)$ as defined in event $\mathsf{WIN1}$, then it can be used in a forgeability attack in the $\mathsf{EUF\text{-}CMA}^{\mathsf{DS}}$ experiment.

Let q denote the number of queries to $\mathsf{QRandKey}$. The adversary $\mathcal{B}$ behaves exactly as described in the experiment, with the only difference that it randomly samples values $i \leftarrow [q], j \leftarrow [\ell]$, where q denotes the number of queries to $\mathsf{QKeyGen}$ and ℓ denotes the number of queries to $\mathsf{QRandKey}$, and, to reply to the j'th $\mathsf{QRandKey}$ query of the i'th key, it uses the key vk obtained from its underlying challenger. If later a signature query is being asked for the j'th rerandomization of the i'th key, then the adversary $\mathcal{B}$ relies on the signing oracle of its underlying challenger to generate the final signature. In case that the i'th key is being corrupted, the adversary $\mathcal{B}$ aborts.

Finally, when $\mathcal{A}$ terminates with output $(\mathsf{pk}^*_S := (\mathsf{ID}^*_S, \mathsf{vk}^*_S, \mathsf{ct}^*_S, \pi^*_S), \mathsf{pk}^*_R := (\mathsf{ID}^*_R, \mathsf{vk}^*_R, \mathsf{ct}^*_R, \pi^*_R), m^*, \sigma^* := (\pi^*, \sigma'))$ check the conditions of $\mathsf{WIN1}$ and check furthermore that the forgery output by $\mathcal{A}$ corresponds to the j'th rerandomization of the i'th key. If this is not the case, $\mathcal{B}$ aborts. If both of the conditions are satisfied, the adversary $\mathcal{B}$ outputs $((m^*, \mathsf{pk}^*_R, \pi^*), \sigma')$ as its forgery to the underlying $\mathsf{EUF\text{-}CMA}^{\mathsf{DS}}$ experiment.

To analyze the above reduction, we need to calculate the probability with which the adversary $\mathcal{B}$ succeeds with the advantage of $\mathcal{A}$. This happens with probability $\frac{1}{q\ell}$, since the adversary $\mathcal{B}$ needs to guess the correct key i that is used by the adversary $\mathcal{A}$ in the forgery, as well as the correct rerandomization j. Since $q\ell$ is polynomial in the security parameter, the lemma follows. $\square$

Lemma 5. *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *be an* $\mathsf{EUF\text{-}CMA}$*-secure signature scheme and* $\mathsf{NIZK}_{\mathcal{L}_2} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is a knowledge sound proof system for* $\mathcal{L}_2$*, then* $\Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}] = \mathsf{negl}(\lambda)$.

Proof. On a high-level, in this setting, 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 $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}_{\mathrm{Hyb}}$, which is identical to $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}$ except that we replace $\mathsf{NIZK}_{\mathcal{L}_2}.\mathsf{Setup}(1^\lambda)$ by the CRS simulation algorithm Ext_1 associated to the NIZK scheme which also outputs the state $\mathsf{st}_{\mathsf{Sign}}$ for the second extraction algorithm Ext_2. It follows directly from the knowledge soundness property of the NIZK, using a standard reduction, that

$$\begin{aligned}\Pr[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}] \\ \leq \Pr_{\mathrm{Hyb}}\left[\mathsf{WIN2}_{\mathcal{A}} \cap \overline{\mathsf{KeyColl}_{\mathcal{A}} \cup \mathsf{KeyForge}_{\mathcal{A}}}\right] + \mathsf{negl}(\lambda),\end{aligned}$$

where $\Pr_{\mathrm{Hyb}}[.]$ makes explicit that this probability is taken w.r.t. the experiment $\mathsf{EUF\text{-}CMA}^{\mathsf{ULPCS}}_{\mathrm{Hyb}}$.

Now, to bound the probability of the occurrence of $\mathsf{KeyForge}$, we need to bound two different subcases:

1. The adversary is not able to forge a signature σ^3_{sig} that would suffice as a proof for the relation $R_{\mathcal{L}_2}$.
2. The adversary is not able to break the soundness of the underlying $\mathsf{NIZK}_{\mathcal{L}_2}$ to generate a valid proof without being in possession of a witness.

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

The adversary $\mathcal{B}$ behaves using as described in the protocol with the only difference that it does not generate the key pair $(\mathsf{vk}^A_{\mathsf{sig}}, \mathsf{sk}^A_{\mathsf{sig}})$ on its own but obtains it from an underlying challenger. Also the corresponding signatures $\sigma^1_{\mathsf{sig}}, \sigma^2_{\mathsf{sig}}$ and σ^3_{sig}, that are the outputs of key generation queries, are not generated by $\mathcal{B}$ directly but through signing oracle queries of $\mathcal{B}$ to its underlying challenger.

When $\mathcal{A}$ terminates with $(\mathsf{pk}^*_S := (\mathsf{ID}^*_S, \mathsf{vk}^*_S, \mathsf{ct}^*_S, \pi^*_S), \mathsf{pk}^*_R := (\mathsf{ID}^*_R, \mathsf{vk}^*_R, \mathsf{ct}^*_R, \pi^*_R), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}_1$ first checks whether the conditions of event $\mathsf{WIN2}$ are fulfilled (and $\mathsf{KeyForge}_{\mathcal{A}}$ and $\mathsf{KeyColl}_{\mathcal{A}}$ did not occur), using the detect procedure which will output S' and R'. If the conditions of $\mathsf{WIN2}$ are not fulfilled, then $\mathcal{B}$ aborts.

If the conditions of event $\mathsf{WIN2}$, and not $\mathsf{KeyForge}_{\mathcal{A}}$ and $\mathsf{KeyColl}_{\mathcal{A}}$, are fulfilled, then $\mathcal{B}$ calls $\mathsf{sk}^* \leftarrow \mathsf{Ext}_2(\mathsf{CRS}_{\mathsf{Sign}}, \mathsf{st}_{\mathsf{Sign}}, (\mathsf{ID}^*_S, \mathsf{vk}^A_{\mathsf{sig}}, \mathsf{ct}^*_R), \pi^*)$ and checks whether $(x := (\mathsf{ID}^*_S, \mathsf{ct}^*_R, \mathsf{vk}^A_{\mathsf{sig}}), w := \mathsf{sk}^*) \in R_{\mathcal{L}_2}$ (which is efficiently checkable). Afterwards, $\mathcal{B}$ parses $\mathsf{sk}^* := (\mathsf{usk}^*, \mathsf{ctr}^*, \mathsf{sk}^{*,\mathsf{ctr}}_{\mathsf{sig}})$ and $\mathsf{usk}^* := (\mathsf{k}^*, \mathsf{vk}^*_{\mathsf{sig}}, \mathsf{sk}^*_{\mathsf{sig}}, \sigma^{*,1}_{\mathsf{sig}}, \sigma^{*,2}_{\mathsf{sig}}, \sigma^{*,3}_{\mathsf{sig}}, x^*, \mathsf{sk}^*_{f_x})$, checks if $\mathsf{DS.Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}^*, \mathsf{sk}^*_{f_x}), \sigma^{*,2}_{\mathsf{sig}}) = 1$ and submits the message-signature-pair $((\mathsf{k}^*, \mathsf{sk}^*_{f_x}), \sigma^{*,3}_{\mathsf{sig}})$ to its challenger if it has not been previously output by the signing oracle. Otherwise, it aborts.

If $(x, w) \notin R_{\mathcal{L}_2}$ then abort with failure event $\mathsf{Fail}_{\text{ext}}$. Therefore, since the described reduction is perfect, it holds that the advantage can be reduced to the unforgeability of the underlying signature scheme with probability $\Pr_{\text{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\text{ext}}}\right]$. This, in turn, results in the fact that $\Pr_{\text{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}}\right] = \Pr_{\text{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\text{ext}}}\right] + \Pr_{\text{Hyb}}\left[\mathsf{Fail}_{\text{ext}}\right] + \mathsf{negl}(\lambda)$.

Since a forgery for the underlying $\mathsf{EUF\text{-}CMA}^{\mathsf{DS}}$ experiment only occurs with negligible probability, it follows that $\Pr_{\text{Hyb}}\left[\mathsf{KeyForge}_{\mathcal{A}} \cap \overline{\mathsf{Fail}_{\text{ext}}}\right] = \mathsf{negl}(\lambda)$ and, to conclude the proof, it only remains to show that $\Pr_{\text{Hyb}}\left[\mathsf{Fail}_{\text{ext}}\right] = \mathsf{negl}(\lambda)$. This can be done by relying on the soundness property of the underlying $\mathsf{NIZK}_{\mathcal{L}_2}$ as mentioned in the second of the two cases above.

Our adversary $\mathcal{B}'$ in the case that $\mathsf{Fail}_{\mathsf{Ext}}$ occurs receives as an input the $\mathsf{CRS}_{\mathsf{Sign}}$ and executes the same instructions as $\mathcal{B}$, with the exceptions that it generates $(\mathsf{vk}^{A}_{\mathsf{sig}}, \mathsf{sk}^{A}_{\mathsf{sig}})$ by itself and can use it to generate signatures by itself. In addition, when $\mathcal{A}$ terminates with output $(\mathsf{pk}^*_S := (\mathsf{ID}^*_S, \mathsf{vk}^*_S, \mathsf{ct}^*_S, \pi^*_S), \mathsf{pk}^*_R := (\mathsf{ID}^*_R, \mathsf{vk}^*_R, \mathsf{ct}^*_R, \pi^*_R), m^*, \sigma^* := (\pi^*, \sigma'))$, $\mathcal{B}'$ behaves as $\mathcal{B}$ but does not execute the final steps running the extractor, but instead just outputs $(x := (\mathsf{ID}^*_S, \mathsf{ct}^*_R, \mathsf{vk}^{A}_{\mathsf{sig}}), \pi^*)$ in case the conditions of $\mathsf{WIN2}$, and not $\mathsf{KeyForge}_{\mathcal{A}}$ and $\mathsf{KeyColl}_{\mathcal{A}}$, are satisfied (note that the extractor is run as part of the knowledge soundness experiment). As above, the emulation towards $\mathcal{A}$ is perfect until the point where $\mathcal{B}'$ would abort. This results in the claimed advantage since the event of interest is that the extractor Ext_2 is called precisely on the accepting proof string π^* output by $\mathcal{A}$ which produces a witness w such that $(x, w) \notin R_{\mathcal{L}_2}$. This concludes the proof of the lemma. □

Analysis in the case of Separable & RBAC Policies

Separable Policies. The security proofs for the scheme covering separable policies proceeds exactly in the same way as the proof described above, i.e. in the proof of Lemmas 3 and 5, where the occurence of exactly the same subevents are being bounded. The reason is that we still have the same components, signatures and encryptions, but thanks to the precomputation of $S(x)$ and $R(x)$ we can mimic the PE part of the generic scheme accurately and securely.

RBAC Policies. The proof for the scheme covering RBAC policies has a few differences when bounding the event $\mathsf{KeyForge}_{\mathcal{A}}$ (Lemma 3). Instead of bounding the unforgeability of the signatures σ^1_{sig} and σ^2_{sig} for the PE-based scheme, in the RBAC scheme it is necessary to bound the unforgeability of σ^1_{sig} and invoke the unforgeability of the SEQ scheme to make sure that none of the parties can obtain a different role (akin to re-encryptions of attributes of the generic scheme). This was previously captured within the NIZK, and now, thanks to SEQ, can be verified outside the NIZK. To argue unforgeability now, we first rely on the secure adaptation property of SEQ to argue that the signature generated using $\mathsf{ChgRep}_{\mathcal{R}}$ is indistinguishable from a signature generated using Sign. Afterwards, we can conclude the proof by relying on the unforgeability of the SEQ scheme and the fact that with overwhelming probability, every party is its own equivalence class, which stems from the fact that for each party, the first component of the vector $\vec{M}$ is a randomly sampled group element. For the proof of event $\mathsf{WIN2}$ (Lemma 5), we also need to rely on the weak soundness property of the accumulator to argue that an adversary cannot forge a signature by forging a valid accumulator. To rely on the accumulator soundness, we observe that a party cannot claim to own different roles than the ones it got issued (akin to the signature on the attribute x in the generic scheme).

B.3 Attribute Hiding

In this section, we prove the attribute hiding of our scheme.

Theorem 6. *Let* $T_{\mathsf{Rand}} = \mathsf{poly}(\lambda)$. *If* $\mathsf{PE} = (\mathsf{PE.Setup}, \mathsf{PE.KeyGen}, \mathsf{PE.Enc}, \mathsf{PE.Dec})$ *is a predicate encryption scheme,* $\mathsf{NIZK}_{\mathcal{L}_1} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *is a NIZK proof system for language* $\mathcal{L}_1$*,* $\mathsf{NIZK}_{\mathcal{L}_2} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *is a NIZK proof system for language* $\mathcal{L}_2$ *and* $\mathsf{DS} = (\mathsf{DS.Setup}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$ *an unforgeable signature scheme, then the construction* $\mathsf{ULPCS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figures 12 and 13, is attribute hiding. Namely, for any valid PPT adversary* $\mathcal{A}$*, it holds that* $\mathsf{Adv}^{\mathsf{AH}}_{\mathsf{ULPCS},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

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

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

Game G_1: In this game, we change the behavior of the sign oracle QSign and define a modified sign oracle QSign'. The oracle QSign' is defined as QSign with the difference that it only answers queries for receiver keys that have been honestly generated (keys that have been output by the key generation oracle $\mathsf{QKeyGenLR}_0$ or are an honest rerandomization of these keys, which can be determined using the Detect procedure), for a query (i, pk', m) with $(i, \cdot, \cdot, \cdot, \cdot) \notin \mathcal{QK}$ or $(j, \cdot, \cdot, \cdot, \cdot) \notin \mathcal{QK}$, where $j \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}', \mathcal{QK})$ the sign oracle QSign' outputs $\perp$. The transition from G_0 to G_1 is justified by the bounds on the key forgery event as described in the proof of Theorem 5. We show this transition more formally in Lemma 6.

Game G_2: In this game, we change from an honestly generated $\mathsf{CRS}_{\mathsf{Rand}}$ and honestly generated proofs to a simulated $\mathsf{CRS}_{\mathsf{Rand}}$ and simulated proofs. That is, for the randomization of challenge keys that can never be corrupted, i.e. for the challenge query (x_0, x_1) it holds that $x_0 \neq x_1$, the proof in the randomization for $R_{\mathcal{L}_1}$ is simulated and therefore does not require the attributes used in the witness. Furthermore, we also remove the signatures σ^1_{sig} and σ^2_{sig} from the scheme in this transition. The transition from G_1 to G_2 is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_1}$. We show this transition more formally in Lemma 7.

Game G_3: In this game, we change from an honestly generated $\mathsf{CRS}_{\mathsf{Sign}}$ and honestly generated proofs to a simulated $\mathsf{CRS}_{\mathsf{Sign}}$ and simulated proofs. That is, upon a signing query we check, from the transcript of the generated keys and using the detect function, if the requested key pair in the signing query fulfills the policy. If this is the case, the proof π_s is simulated using $\mathsf{CRS}_{\mathsf{Sign}}$. Here, we furthermore also remove the key sk_{f_x} as well as the signature σ^3_{sig} from the key generation procedure. As in the previous transition, this also only happens for explicitly honest keys, i.e. keys where $x_0 \neq x_1$. The transition from G_2 to G_3 is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_2}$. We show this transition more formally in Lemma 8.

Game G_4: In this game, we change the attributes used in the rerandomization for the explicitly honest challenge keys from x_0 to x_1 for all i by changing the encryption that is being generated in the randomization procedure. The transition from G_3 to G_4 is justified by the attribute-hiding property of PE. We show this transition more formally in Lemma 9.

Game G_5: In this game, we change back from a simulated $\mathsf{CRS}_{\mathsf{Sign}}$ and simulated proofs to an honestly generated $\mathsf{CRS}_{\mathsf{Sign}}$ and honestly generated proofs. Here, we also reintroduce the signature σ^3_{sig} but this time w.r.t. the challenge messages x_1. Similar to the transition from G_2 to G_3, this transition is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_2}$.

Game G_6: In this game, we change back from a simulated $\mathsf{CRS}_{\mathsf{Sign}}$ and simulated proofs to an honestly generated $\mathsf{CRS}_{\mathsf{Sign}}$ and honestly generated proofs. Here, we also reintroduce the signatures σ^1_{sig} and σ^2_{sig} but this time w.r.t. the challenge messages x_1. Similar to the transition from G_1 to G_2, this transition is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_1}$.

Game G_7: This game is the $\mathsf{AH}_1^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A})$ game. In this game, we change the behavior of the signing oracle back from QSign' to QSign. Similar to the transition from G_0 to G_1, this transition is justified by the event $\mathsf{KeyForge}_{\mathcal{A}}$.

From the definition of the games it is clear that

$$\mathsf{AH}_0^{\mathsf{ULPCS}} = G_0 \approx G_1 \approx \cdots \approx G_7 = \mathsf{AH}_1^{\mathsf{ULPCS}}$$

and hence the theorem follows. □

Lemma 6 (Transition from G_0 to G_1). *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 the game G_0 the adversary $\mathcal{A}$ has access to the sign oracle QSign and in the game G_1 the adversary $\mathcal{A}$ has access to the sign oracle QSign', which we informally described above and which is formally defined as:

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

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

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

For the event $\mathsf{SignForge}_{\mathcal{A}}$ to occur, the adversary $\mathcal{A}$ needs to generate a receiver key that has a valid zero-knowledge proof π, or where the underlying witness is forged, a valid signature $\mathsf{sig}^1_{\mathsf{sig}}$ the signature scheme DS, i.e., it needs to generate a key $\mathsf{pk}' := (\mathsf{ID}', \mathsf{vk}', c', \pi)$ such that $\mathsf{NIZK}_{\mathcal{L}_1}.\mathsf{Verify}(\mathsf{CRS}_{\mathsf{Rand}}, (\mathsf{ID}', \mathsf{vk}', c', \mathsf{vk}^A_{\mathsf{sig}}), \pi) = 1$ where π is generated using $\mathsf{usk}' := (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{PE}}, \sigma^1_{\mathsf{sig}}, \sigma^2_{\mathsf{sig}}, \sigma^3_{\mathsf{sig}}, x)$ and it must holds that $\mathsf{DS}.\mathsf{Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, x), \sigma^1_{\mathsf{sig}}) = 1$ and $\mathsf{DS}.\mathsf{Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}), \sigma^2_{\mathsf{sig}}) = 1$. This means that adversary $\mathcal{A}$ must either break the soundness of NIZK or generate a key forgery for DS as captured by the event $\mathsf{KeyForge}_{\mathcal{A}}$ in the proof of Theorem 5, and which can be defined and analyzed analogously here.

Therefore, the event $\mathsf{SignForge}_{\mathcal{A}}$ is bounded by $\mathsf{KeyForge}$, i.e. $\Pr[\mathsf{SignForge}_{\mathcal{A}}] \leq \Pr[\mathsf{KeyForge}_{\mathcal{A}}]$, and the analysis of event $\mathsf{KeyForge}_{\mathcal{A}}$ follows the same reasoning as in Lemma 3. This results in the fact that $\Pr[\mathsf{SignForge}_{\mathcal{A}}] = \mathsf{negl}(\lambda)$, which proves the lemma. □

Lemma 7 (Transition from G_1 to G_2). *The games G_1 and G_2 are computationally indistinguishable.*

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

The adversary $\mathcal{B}$ behaves in the same way as described in G_1 with the difference that it does not generate $\mathsf{CRS}_{\mathsf{Rand}}$ by itself but receives it from the underlying challenger. Additionally, whenever the adversary $\mathcal{A}$ asks a rerandomization query to $\mathsf{QRandKey}$ for a key that cannot be corrupted, i.e. where the key generation query is for $x_0 \neq x_1$, the adversary $\mathcal{B}$ behaves as described in the protocol but uses the proof oracle of the challenger for the generation of the proof π_{k+1}. Furthermore, the signature σ^1_{sig} and σ^2_{sig} are not generated.

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

To conclude the proof, we argue that our emulation is perfect. The fact that the simulation is perfect follows since $\mathcal{B}$ generates all components of the statement for which the proof oracle is queried honestly.

In the case that the challenger outputs an honestly generated $\mathsf{CRS}_{\mathsf{Rand}}$ and honestly generated proofs, the adversary $\mathcal{B}$ is simulating the game G_1 and in the case that the challenger simulates the $\mathsf{CRS}_{\mathsf{Rand}}$ and the proofs, the adversary $\mathcal{B}$ is simulating the game G_2.

This concludes the simulation of the game $G_{1+\beta}$ and the lemma follows. □

Lemma 8 (Transition from G_2 to G_3). *The games G_2 and G_3 are computationally indistinguishable.*

Proof. We build an adversary $\mathcal{B}$ that simulates $G_{2+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\mathsf{ZK}^{\mathsf{NIZK}}_{\beta}$ experiment.

The adversary $\mathcal{B}$ behaves in the same way as described in G_2 with the difference that it does not generate $\mathsf{CRS_{Sign}}$ by itself but receives it from the underlying challenger. Additionally, whenever the adversary $\mathcal{A}$ asks a signing query (i, pk', m) to QSign', the adversary $\mathcal{B}$ computes $j \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}', \mathcal{QK})$ and checks that $F(x_0, y_0) = 1$ where $(i, \cdot, \cdot, x_0, x_1) \in \mathcal{QK}$ and $(j, \cdot, \cdot, y_0, y_1) \in \mathcal{QK}$. If the check succeeds, then $\mathcal{B}$ queries its underlying proof oracle to obtain π_s and finishes the signature generation. Furthermore, for all keys that cannot be corrupted, i.e. where the key generation query is for $x_0 \neq x_1$, the signature σ^3_{sig} is not generated and the key sk_{f_x} is not generated. This makes the secret key completely independent of the attributes x_0/x_1.

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

To conclude the proof, we argue that our emulation is perfect. The fact that the simulation is perfect follows since $\mathcal{B}$ only submits proof queries to the underlying challenger for which the statement fulfills the relation $R_{\mathcal{L}_2}$, which $\mathcal{B}$ checks as described above as well as from the perfect correctness of the predicate encryption scheme. In more detail, by the perfect correctness of the predicate encryption scheme, we know that the challenger always replies, i.e., we have that $\mathsf{PE.Dec}(\mathsf{sk}_{f_x}, \mathsf{ct}_R) = F(x, y)$. Therefore, whenever a proof is simulated this matches the correct generation of a proof π_s.

In the case that the challenger outputs an honestly generated $\mathsf{CRS_{Sign}}$ and honestly generated proofs, the adversary $\mathcal{B}$ is simulating the game G_2 and in the case that the challenger simulates the $\mathsf{CRS_{Sign}}$ and the proofs, the adversary $\mathcal{B}$ is simulating the game G_3.

This concludes the simulation of the game $G_{2+\beta}$ and the lemma follows. □

Lemma 9 (Transition from G_3 to G_4). *The games G_3 and G_4 are computationally indistinguishable.*

Proof. We build an adversary $\mathcal{B}$ that simulates $G_{3+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\mathsf{AH}^{\mathsf{PE}}_{\beta}$ experiment.

The adversary $\mathcal{B}$ behaves in the same way as described in G_3 with the difference that whenever the adversary $\mathcal{A}$ asks a key generation query for a key that can be corrupted, i.e. $x \coloneqq x_0 = x_1$, the adversary $\mathcal{B}$ asks its underlying key generation oracle using x to obtain sk_{f_x}.

Additionally, when $\mathcal{A}$ asks a rerandomization query to $\mathsf{QRandKey}$ for a key that cannot be corrupted, i.e. where the key generation query is for $x_0 \neq x_1$, the adversary $\mathcal{B}$ behaves as described in the protocol but uses its underlying left-or-right oracle for the generation of the ciphertext, i.e. for every rerandomization query for a key i, $\mathcal{B}$ retrieves $(i, \cdot, \cdot, x_0, x_1) \in \mathcal{QK}$ and submits (x_0, x_1) to its underlying challenger to obtain ct which it uses for the rerandomization.

Furthermore, for every sign query (j, pk_R, m) to QSign' asked by $\mathcal{A}$, $\mathcal{B}$ computes $j \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}', \mathcal{QK})$ checks the list $\mathcal{QK}$ to find $(i, \cdot, \cdot, x_0, x_1)$ and $(j, \cdot, \cdot, y_0, y_1)$. If no such entries exists, $\mathcal{B}$ outputs $\perp$. Otherwise, $\mathcal{B}$ checks that that the attributes associated with the public keys pk_S and pk_R fulfill the policy, i.e. it checks that $F(x^0, y^0) = 1$ and $F(x^1, y^1) = 1$, and if this is the case simulates the proof and generates the signature.

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

In the next step, we need to argue that the adversary $\mathcal{B}$ is a valid adversary with respect to the $\mathsf{AH}_\beta^{\mathsf{PE}}$ experiment if the adversary $\mathcal{A}$ fulfills all the checks described above, i.e. is a valid adversary in the $G_{3+\beta}$ ($\mathsf{AH}_\beta^{\mathsf{ULPCS}}$) game. One of the validity requirements above (and in the attribute hiding game) that $\mathcal{A}$ needs to fulfill is that for every x where $x \coloneqq x_0 = x_1$ with $(\cdot,\cdot,\cdot,x_0,x_1) \in \mathcal{QS}$ it needs to hold that $F(x,x_0) = F(x,x_1)$ for all the challenge queries (x_0,x_1). This results in the fact that $f_x(x_0) = f_x(x_1)$ for all $(\cdot,\cdot,\cdot,x,x) \in \mathcal{QC}$ and for all challenge queries (x_0,x_1). This matches exactly the validity requirements asked for $\mathcal{B}_2$ in the $\mathsf{AH}_\beta^{\mathsf{PE}}$ experiment. Therefore, it follows that the adversary $\mathcal{B}_2$ is a valid adversary with respect to the $\mathsf{AH}_\beta^{\mathsf{PE}}$ experiment and does not abort if the adversary $\mathcal{A}$ is a valid adversary in the game $G_{2+\beta}$ ($\mathsf{AH}_\beta^{\mathsf{ULPCS}}$).

To conclude the proof, we observe that the difference in the two games is the generation of the challenge rerandomization keys, 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 $G_{3+\beta}$ 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 $G_{3+\beta}$ and the lemma follows. $\square$

Analysis in the case of Separable & RBAC Policies

Separable Policies. The security proof for the scheme covering separable policies proceeds in almost the same way as the proof for general policies. The only difference is the transition from G_3 to G_4 (Lemma 9), where in the proof for separable policies we need to rely on the IND-CPA security of the underlying public-key encryption scheme PKE instead of the attribute-hiding security of a PKE scheme.

RBAC Policies. For the security proof of the scheme covering RBAC policies, we also need to adjust the transition from game G_3 to G_4 (Lemma 9). In this case, we need to rely on the class-hiding property as well as the secure adaptation property. In more detail, the class-hiding property guarantees that a switch from attributes x_0 to x_1 (in the case of outsider attribute-hiding or in the case of the equality policy) is possible and the secure adaptation property of SEQ ensures that the $\mathsf{ChgRep}_\mathcal{R}$ algorithm is as good as re-generating $\vec{M}$, which fulfills the same purpose as the re-encryption for the schemes covering general and separable policies.

B.4 Unlinkability

This section, covers the unlinkability proof of our schemes.

Theorem 7. *Let* $T_{\mathsf{Rand}} = \mathsf{poly}(\lambda)$. *If* PRF *is a pseudorandom function,* $\mathsf{NIZK}_{\mathcal{L}_1} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *a NIZK proof system for* $\mathcal{L}_1$, $\mathsf{NIZK}_{\mathcal{L}_1} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *a NIZK proof system for* $\mathcal{L}_1$ *and* $\mathsf{DS} = (\mathsf{DS.Setup}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$ *an unforgeable signature scheme, then the construction* $\mathsf{ULPCS} = (\mathsf{Setup},$

$\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined in Figures 12 and 13, is unlinkable. Namely, for any valid PPT adversary* $\mathcal{A}$*, it holds that* $\mathsf{Adv}^{\mathsf{Link}}_{\mathsf{ULPCS},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$*.*

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

Game G_0**:** This game is the same as the experiment $\mathsf{Link}^{\mathsf{ULPCS}}_0(1^\lambda, \mathcal{A})$.

Game G_1**:** In this game, we change the behavior of the key generation oracle $\mathsf{QKeyGen}$ and define a modified key generation oracle $\mathsf{QKeyGen}'$. The oracle $\mathsf{QKeyGen}'$ is defined as $\mathsf{QKeyGen}$ with the difference that it does not output a key collision, i.e. it does not output the same public key twice, and therefore does also not output the same public key as the challenge key. More formally, if for a query x', the output is $\mathsf{pk}' := (\mathsf{ID}', \dots)$ where $\mathcal{QK}$ already contains $(\dots, \mathsf{pk}^* := (\mathsf{ID}^*, \dots), \dots)$ with $\mathsf{ID}' = \mathsf{ID}^*$ or $\mathsf{ID}' = \mathsf{ID}'$ with $\mathsf{pk} := (\mathsf{ID}, \dots)$ being the challenge public key, then the key-generation oracle $\mathsf{QKeyGen}'$ outptus $\perp$, otherwise it returns pk'. The transition from G_0 to G_1 is justified by the bounds on the key collision event as described in the proof of Theorem 5. We show this transition more formally in Lemma 10.

Game G_2**:** In this game, we change the behavior of the sign oracle QSign and define a modified sign oracle QSign'. As in the proof of Theorem 6, the oracle QSign' is defined as QSign with the difference that it only answers queries for receiver keys that it can detect to have come out of the key-gen oracle. For further details on QSign', we refer to the proof of Theorem 6. The transition from G_1 to G_2 is justified by the bounds on the key forgery event as described in the proof of Theorem 5 and because the detect property is fulfilled by the scheme. This transition has been shown in Lemma 6.

Game G_3**:** In this game, we change from an honestly generated $\mathsf{CRS}_{\mathsf{Rand}}$ and honestly generated proofs w.r.t. rerandomizations of the challenge public key pk to a simulated $\mathsf{CRS}_{\mathsf{Rand}}$ and simulated proofs for the rerandomizations of pk. Due to the fact that the proofs for the rerandomizations are now simulated, the PRF key k is not needed as part of the witness anymore. The transition from G_2 to G_3 is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_1}$. We show this transition more formally in Lemma 11.

Game G_4**:** In this game, we change from an honestly generated $\mathsf{CRS}_{\mathsf{Sign}}$ and honestly generated proofs for signing queries w.r.t. the challenge public key pk acting as the sender to a simulated $\mathsf{CRS}_{\mathsf{Sign}}$ and simulated proofs. That is, upon a signing query for pk, acting as the sender, we check, from the transcript of the generated keys and using the detect function, if the requested key pair in the signing query fulfills the policy. If this is the case, the proof π_s is simulated using $\mathsf{CRS}_{\mathsf{Sign}}$. Since the proof π_s is now simulated, the PRF key k is not needed as part of the witness anymore. The transition from G_3 to G_4 is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_2}$. We show this transition more formally in Lemma 12.

Game G_5**:** In this game, we change from PRF evaluations for the updated ID's in the rerandomization of the challenge key pk to randomly sampled ID's. The transition from G_4 to G_5 is justified by the security of the PRF. We show this transition more formally in Lemma 13.

Game G_6 : In this game, we change from randomly sampled updated ID's in the rerandomization of the challenge key pk to PRF evaluations w.r.t. different keys. In more detail, whenever a new rerandomization for the challenge key pk is generated a new PRF key k_i is sampled and the ID is generated by evaluation PRF using k_i on 0. The transition from G_5 to G_6 is justified by relying on the security of the PRF q-times where q is the number of rereandomization queries. Since it holds that $q < T_{\mathsf{Rand}}$, we can upper bound it by relying on the security of the PRF T_{Rand} times. We show this transition more formally in Lemma 14.

Game G_7**:** In this game, we change back from a simulated $\mathsf{CRS}_{\mathsf{Sign}}$ and simulated proofs to an honestly generated $\mathsf{CRS}_{\mathsf{Sign}}$ and honestly generated proofs. Here, we also reintroduce the usage of the PRF key k, which is different for every rerandomization, into the generation of the proof. Similar to the transition from G_3 to G_4, this transition is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_2}$.

Game G_8**:** In this game, we change back from a simulated $\mathsf{CRS}_{\mathsf{Sign}}$ and simulated proofs to an honestly generated $\mathsf{CRS}_{\mathsf{Sign}}$ and honestly generated proofs. Here, we also reintroduce the usage of the PRF key k, which is different for every ID, into the generation of the proof. Similar to the transition from G_2 to G_3, this transition is justified by the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{L}_1}$.

Game G_9**:** In this game, we change the behavior of the signing oracle back from QSign' to QSign. Similar to the transition from G_1 to G_2, this transition is justified by the event $\mathsf{KeyForge}_{\mathcal{A}}$.

Game G_{10}**:** This game is the $\mathsf{Link}_1^{\mathsf{ULPCS}}(1^\lambda, \mathcal{A})$ game. In this game, we change the behavior of the key generation oracle back from $\mathsf{QKeyGen}'$ to $\mathsf{QKeyGen}$. Similar to the transition from G_0 to G_1, this transition is justified by the event $\mathsf{KeyColl}_{\mathcal{A}}$.

From the definition of the games it is clear that

$$\mathsf{Link}_0^{\mathsf{ULPCS}} = G_0 \approx G_1 \approx \cdots \approx G_{10} = \mathsf{Link}_1^{\mathsf{ULPCS}}$$

and hence the theorem follows. □

Lemma 10 (Transition from G_0 **to** G_1**).** *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 the game G_0 the adversar $\mathcal{A}$ has access to the key generation oracle $\mathsf{QKeyGen}$ and in the game G_1 the adversary $\mathcal{A}$ has access to the key generation oracle $\mathsf{QKeyGen}'$, which we informally described above and which is formally defined as:

$\mathsf{QKeyGen}'(x')$**:** On input an attribute set x', generate $\mathsf{pk}' := (\mathsf{ID}', \dots)$ and if $\mathcal{QK}$ already contains an entry $(\dots, \mathsf{pk}^* := (\mathsf{ID}^*, \dots), \dots)$ with $\mathsf{ID}' = \mathsf{ID}^*$ or if $\mathsf{ID}' = \mathsf{ID}$ where $\mathsf{pk} := (\mathsf{ID}, \dots)$ is the challenge public key, then output $\perp$. Otherwise, return pk'.

Compared to the oracle $\mathsf{QKeyGen}'$, the key generation oracle $\mathsf{QKeyGen}$ does not require a generated public pk' to be entirely new, i.e. $(\dots, \mathsf{pk}^* := (\mathsf{ID}^*, \dots), \dots) \notin \mathcal{QK}$ with $\mathsf{ID} \neq \mathsf{ID}^*$

and $\mathsf{ID}' \neq \mathsf{ID}$ where $\mathsf{pk} := (\mathsf{ID}, \dots)$ is the challenge public key. To show that the games G_0 and G_1 are indistinguishable, it suffices to show that the probability that two honestly generated IDs do not collide is negligible. This directly matches the description of the event $\mathsf{KeyColl}_{\mathcal{A}}$ defined in the proof of Theorem 5 and, since the it holds that $\Pr[\mathsf{KeyColl}_{\mathcal{A}}] = \mathsf{negl}(\lambda)$, the lemma follows. □

Lemma 11 (Transition from G_2 to G_3). *The games G_2 and G_3 are computationally indistinguishable.*

Proof. This proof is very similar to the proof of Lemma 7.

We build an adversary $\mathcal{B}$ that simulates $G_{2+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\mathsf{ZK}_{\beta}^{\mathsf{NIZK}}$ experiment.

The adversary $\mathcal{B}$ behaves in the same way as described in G_2 with the difference that it does not generate $\mathsf{CRS}_{\mathsf{Rand}}$ by itself but receives it from the underlying challenger. Additionally, whenever the adversary $\mathcal{A}$ asks a rerandomization query to $\mathsf{QRandKey}$ for the challenge public key pk, or a rerandomization of it, the adversary $\mathcal{B}$ behaves as described in the protocol but uses the proof oracle of the challenger for the generation of the proof π_{k+1}. Furthermore, the PRF key k is not used as a witness for the proof generation anymore.

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

To conclude the proof, we argue that our emulation is perfect. The fact that the simulation is perfect follows since $\mathcal{B}$ generates all components of the statement for which the proof oracle is queried honestly.

In the case that the challenger outputs an honestly generated $\mathsf{CRS}_{\mathsf{Rand}}$ and honestly generated proofs, the adversary $\mathcal{B}$ is simulating the game G_2 and in the case that the challenger simulates the $\mathsf{CRS}_{\mathsf{Rand}}$ and the proofs, the adversary $\mathcal{B}$ is simulating the game G_3.

This covers the simulation of the game $G_{2+\beta}$ and leads to the advantage mentioned in the lemma. □

Lemma 12 (Transition from G_3 to G_4). *The games G_3 and G_4 are computationally indistinguishable.*

Proof. This proof is very similar to the proof of Lemma 8.

We build an adversary $\mathcal{B}$ that simulates $G_{3+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\mathsf{ZK}_{\beta}^{\mathsf{NIZK}}$ experiment.

The adversary $\mathcal{B}$ behaves in the same way as described in G_3 with the difference that it does not generate $\mathsf{CRS}_{\mathsf{Sign}}$ by itself but receives it from the underlying challenger. Additionally, whenever the adversary $\mathcal{A}$ asks a signing query (pk', m) to QSign', the adversary $\mathcal{B}$ computes $j \leftarrow \mathsf{Detect}(\mathsf{mpk}, \mathsf{pk}', \mathcal{QK})$ and checks that $F(x, y) = 1$ where x is the challenge attribute and $(j, \cdot, \cdot, y) \in \mathcal{QK}$. If the check succeeds, then $\mathcal{B}$ queries its underlying proof oracle to obtain π_s and finishes the signature generation. Furthermore, the PRF key k is not needed for the generation of the proof π_s.

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

To conclude the proof, we argue that our emulation is perfect. The fact that the simulation is perfect follows since $\mathcal{B}$ only submits proof queries to the underlying challenger for which the statement fulfills the relation $R_{\mathcal{L}_2}$, which $\mathcal{B}$ checks as described above as well as from the perfect correctness of the predicate encryption scheme. In more detail, by the perfect correctness of the predicate encryption scheme, we know that the challenger always replies, i.e., we have that $\mathsf{PE.Dec}(\mathsf{sk}_{f_x}, \mathsf{ct}_R) = F(x, y)$. Therefore, whenever a proof is simulated this matches the correct generation of a proof π_s.

In the case that the challenger outputs an honestly generated $\mathsf{CRS}_{\mathsf{Sign}}$ and honestly generated proofs, the adversary $\mathcal{B}$ is simulating the game G_3 and in the case that the challenger simulates the $\mathsf{CRS}_{\mathsf{Sign}}$ and the proofs, the adversary $\mathcal{B}$ is simulating the game G_4.

This covers the simulation of the game $G_{3+\beta}$ and leads to the advantage mentioned in the lemma. □

Lemma 13 (Transition from G_4 to G_5). *The games G_4 and G_5 are computationally indistinguishable.*

Proof. We build an adversary $\mathcal{B}$ that simulates $G_{4+\beta}$ towards $\mathcal{A}$ when interacting with the underlying security experiment for PRF.

The adversary $\mathcal{B}$ behaves in the same way as described in G_4 with the difference that whenever the adversary $\mathcal{A}$ asks the challenge key generation query or a rereandomization query, the adversary $\mathcal{B}$ submits the corresponding index, i.e. $i := 0$ for a key generation query and $i := i + 1$ for a rerandomizaton query, to the underlying PRF challenger and receives as a reply the ID that it uses to answer the queries.

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

To conclude the proof, we observe that the difference in the two games is the generation of the ID's, which is either PRF evaluation, in which case the simulation corresponds to game G_4, or a random value, in which case the simulation corresponds to game G_5.

This concludes the simulation of the game $G_{4+\beta}$ and the lemma follows. □

Lemma 14 (Transition from G_5 to G_6). *The games G_5 and G_6 are computationally indistinguishable.*

Proof. We build an adversary $\mathcal{B}$ that simulates $G_{5+\beta}$ towards $\mathcal{A}$ when interacting with T instances[9] of the security experiment for PRF.

The adversary $\mathcal{B}$ behaves in the same way as described in G_5 with the difference that whenever the adversary $\mathcal{A}$ asks the challenge key generation query or a rereandomization query, the adversary $\mathcal{B}$ submits the 0 query to the i'th PRF instance. In more detail, for the challenge key generation query, the adversary $\mathcal{B}$ queries the first instance of the PRF experiment on 0 ot obtain the ID, for the first rerandomization query, the adversary $\mathcal{B}$ queries the second instance of the PRF experiment on 0 to obtain the ID and so on.

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

To conclude the proof, we observe that the difference in the two games is the generation of the ID's, which is either a random value, in which case the simulation corresponds to game G_4, or a fresh PRF evaluation on zero, in which case the simulation corresponds to game G_5.

This concludes the simulation of the game $G_{4+\beta}$ and the lemma follows. □

[9] where T instances means multiple PRF experiments that either all output random values or all PRF evaluations on a fresh key

Analysis in the case of Separable & RBAC Policies

Separable Policies. The security proof for the scheme covering separable policies proceeds in the same way as the proof for general policies. In both of these cases, general policies and separable policies, it is not necessary to rely on any of the properties of the predicate encryption scheme or the public-key encryption scheme since, in both cases, rerandomization and key generation, a fresh ciphertext is being generated. Therefore no indistinguishability needs to be argued.

RBAC Policies. In the RBAC case, there is a difference between the rerandomization and the key generation w.r.t. the SEQ scheme. In the case of a fresh key generation, a signature is generated on which $\mathsf{ChgRep}_{\mathcal{R}}$ is applied once. In the case of an actual rerandomization, the $\mathsf{ChgRep}_{\mathcal{R}}$ is applied multiple times on a single signature. To argue that these two cases are indistinguishable, we need to rely on the secure adaptation property of SEQ which can be understood as an additional game transition between G_5 and G_6 (Lemma 14).

C Details on the Instantiations

C.1 Cryptographic Algorithms

Given the formal definitions from Section 3, this section provides a detailed overview on the underlying cryptographic tools that we implement to realize ul-PCS.

Dodis-Yampolskiy PRF. This PRF is defined over a cyclic group $\mathbb{G}$ of prime order p with generator G and can be described as follows:

- $\mathsf{PRF.Eval}(\mathsf{k}, x)$: It takes a key $\mathsf{k} \in \mathbb{Z}_p$ and input x as inputs. It then computes $y = \mathsf{G}^{1/(\mathsf{k}+x)}$ and returns y as output.

In this scheme, pseudo-randomness is achieved through decisional Diffie-Hellman inversion assumption, which only holds for small domains, i.e. input x should be super-logarithmic in the security parameters.

BLS Signatures. For a given assymetric bilinear pairing group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathsf{G}_1, \mathsf{G}_2)$ and a hash-to-curve function $H : \{0,1\}^* \rightarrow \mathbb{G}_2$, as denoted by public parameters pp, we recall the BLS signatures [15] as follows:

- $\mathsf{DS.Setup}(1^\lambda)$: Take pp as input. Sample $x \xleftarrow{\$} \mathbb{Z}_p^*$. Return $(\mathsf{sk}, \mathsf{vk}) = (x, \mathsf{G}_1^x)$.
- $\mathsf{DS.Sign}(\mathsf{sk}, m)$: Take secret key sk and message $m \in \{0,1\}^*$ as inputs. Return $\sigma := H(m)^{\mathsf{sk}}$ as output.
- $\mathsf{DS.Verify}(\mathsf{vk}, \sigma, m)$: Take the verification key vk, a signature σ and message m as inputs. If the equation $e(\mathsf{G}_1, \sigma) = e(\mathsf{vk}, H(m))$ holds, return 1 and 0 otherwise.

FHS SPS-EQ. We recall the SPS-EQ construction proposed by Fuchsbauer et al. in [37] as follows:

- $\mathsf{SEQ.Setup}_{\mathcal{R}}(1^\lambda)$: Run $\mathsf{BG} \leftarrow \mathcal{BG}(1^\lambda)$ and return $\mathsf{pp} := \mathsf{BG}$ as output.
- $\mathsf{SEQ.KeyGen}_{\mathcal{R}}(\mathsf{pp}, \ell)$: Take pp and vector size $\ell > 1$ as inputs. Sample the secret key sk as a set of random integers $\mathsf{sk} := \{x_i\}_{i\in[1,\ell]} \xleftarrow{\$} (\mathbb{Z}_p^*)^\ell$. Compute $\mathsf{vk} := \{\hat{X}_i = \mathsf{G}_1^{x_i}\}_{i\in[1,\ell]}$. Return $(\mathsf{sk}, \mathsf{vk})$ as output.
- $\mathsf{SEQ.Sign}_{\mathcal{R}}(\mathsf{pp}, \mathsf{sk}, \vec{M})$: Parse $\vec{M} := (M_i)_{i\in[1,\ell]} \in (\mathbb{G}_2)^\ell$ and $\mathsf{sk} : \{x_i\}_{i\in[1,\ell]}$. Sample $a \xleftarrow{\$} \mathbb{Z}_p^*$ and return $\sigma := (R, S, T) := \left(\left(\prod_{i\in[1,\ell]} M_i^{x_i}\right)^a, \mathsf{G}_2^{1/a}, \mathsf{G}_1^{1/a}\right) \in \mathbb{G}_2^2 \times \mathbb{G}_1$ as output.
- $\mathsf{SEQ.Verify}_{\mathcal{R}}(\mathsf{pp}, \mathsf{vk}, \vec{M}, \sigma)$: Parse $\mathsf{vk} := \{\hat{X}_i\}_{i\in[1,\ell]}$, $\vec{M} := (M_i)_{i\in[1,\ell]}$ and $\sigma := (R, S, T)$. If the equations $\prod_{i\in[1,\ell]} e(\hat{X}_i, M_i) = e(T, R)$ and $e(\mathsf{G}_1, S) = e(T, \mathsf{G}_2)$ hold and $M_i \neq 1_{\mathbb{G}_2}$ for $i \in [1, \ell]$ return 1 and 0 otherwise.
- $\mathsf{SEQ.ChgRep}_{\mathcal{R}}(\mathsf{pp}, \vec{M}, \sigma, \mu, \mathsf{vk})$: Parse $\sigma := (R, S, T)$, $\vec{M} := (M_i)_{i\in[1,\ell]} \in (\mathbb{G}_2)^\ell$ and $\mathsf{vk} := \{\hat{X}_i\}_{i\in[1,\ell]}$ along with an integer $\mu \in \mathbb{Z}_p^*$ as input. If the signature be valid it samples $\zeta \xleftarrow{\$} \mathbb{Z}_p^*$ and then returns $\sigma' := (R', S', T') \leftarrow (R^{\zeta\mu}, S^{1/\zeta}, T^{1/\zeta})$ on a re-randomized message $\vec{M}' = \vec{M}^\mu$ as output.

For simplicity we take a slightly modified variant of the described SPS-EQ as a standard SPS. Consider pairing group of the form $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \mathsf{G}_1, \mathsf{G}_2)$, this scheme can be summarized as follows:

- $\mathsf{SPS.KeyGen}(\mathsf{pp}, \ell)$: Take pp and vector size $\ell > 1$ as inputs. Sample the secret key sk as a set of random integers $\mathsf{sk} := \{x_i\}_{i\in[1,\ell]} \xleftarrow{\$} (\mathbb{Z}_p^*)^\ell$. Compute $\mathsf{vk} := \{\hat{X}_i = \mathsf{G}_2^{x_i}\}_{i\in[1,\ell]}$. Return $(\mathsf{sk}, \mathsf{vk})$ as output.
- $\mathsf{SPS.Sign}(\mathsf{pp}, \mathsf{sk}, \vec{M})$: Parse $\vec{M} := (M_i)_{i\in[1,\ell]} \in \mathbb{G}_1^\ell$ and $\mathsf{sk} := \{x_i\}_{i\in[1,\ell]}$. Sample $a \xleftarrow{\$} \mathbb{Z}_p^*$ and output $\sigma := (R, S, T) := \left(\left(\prod_{i\in[1,\ell]} M_i^{x_i}\right)^a, \mathsf{G}_1^{1/a}, \mathsf{G}_2^{1/a}\right) \in \mathbb{G}_1^2 \times \mathbb{G}_2$.
- $\mathsf{SPS.Verify}(\mathsf{pp}, \mathsf{vk}, \sigma, \vec{M})$: Parse $\mathsf{vk} := \{\hat{X}_i\}_{i\in[1,\ell]} \in \mathbb{G}_2^\ell$, $\vec{M} := (M_i)_{i\in[1,\ell]} \in (\mathbb{G}_1)^\ell$ and $\sigma := (R, S, T)$. If both equations $\prod_{i\in[1,\ell]} e(M_i, \hat{X}_i) = e(R, T)$ and $e(S, \mathsf{G}_2) = e(\mathsf{G}_1, T)$ hold and $M_i \neq 1_{\mathbb{G}_1}$ for $i \in [1, \ell]$ return 1 and 0 otherwise.

ElGamal Encryption. Consider a group description $(\mathbb{G}, \mathsf{G}, p)$, the ElGamal encryption [30] can be formalized as follows:

- $\mathsf{PKE.Setup}(1^\lambda)$: It takes security parameter λ as input and then samples random integer $\mathsf{sk} \leftarrow \mathbb{Z}_p^*$ and computes $\mathsf{pk} = \mathsf{G}^{\mathsf{sk}}$. It then returns the key-pair $(\mathsf{sk}, \mathsf{pk})$ as output.
- $\mathsf{PKE.Enc}(\mathsf{pp}, \mathsf{pk}, m)$: It takes pp, public key pk and message $m \in \mathbb{G}$ as inputs. It samples a random integer $r \xleftarrow{\$} \mathbb{Z}_p^*$ and returns the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (\mathsf{G}^r, m \cdot \mathsf{pk}^r)$ as output.
- $\mathsf{PKE.Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$: It takes pp, the secret key sk and ciphertext ct as inputs. It then returns $m' = \mathsf{ct}_2/(\mathsf{ct}_1)^{\mathsf{sk}}$ as output.

The security of this construction relies on the hardness of DDH assumption over group $\mathbb{G}$. Over a bilinear group, if SXDH holds (DDH is hard in $\mathbb{G}_1$ and $\mathbb{G}_2$), like Type-III bilinear groups, then ElGamal encryption remains secure over source groups $(\mathbb{G}_1, \mathsf{G}_1, p)$ and $(\mathbb{G}_2, \mathsf{G}_2, p)$.

Pedersen Commitment. Commitment schemes enable a committer to commit to a hidden value by ensuring two main security properties: (perfectly) hiding and (computationally) binding. The hiding of the commitment ensures that no information about the hidden committed value is revealed and binding guarantees no committer can open the same commitment under two distinct messages. The Pedersen commitment [56] can be described as follows:

- $\mathsf{COM.Setup}(1^\lambda)$: Take security parameter, λ, as input. Sample $\mathsf{G} \xleftarrow{\$} \mathbb{G}$ and $\mathsf{H} \xleftarrow{\$} \mathbb{G}$. Return the public parameters $\mathsf{pp} = (\mathbb{G}, p, \mathsf{G}, \mathsf{H})$ as output.
- $\mathsf{COM.Com}(\mathsf{pp}, m; \tau)$: Take public parameters pp, a message $m \in \mathbb{Z}_\mathsf{p}$ and random opening τ as inputs. Output $\mathsf{cm} = \mathsf{G}^m \mathsf{H}^\tau$.
- $\mathsf{COM.Verify}(\mathsf{pp}, \mathsf{cm}, m', \tau')$: Compute $\mathsf{cm}' = \mathsf{G}^{m'} \mathsf{H}^{\tau'}$. Return 1, if $\mathsf{cm} = \mathsf{cm}'$; otherwise return 0.

Generalized Pedersen Commitments. The Pedersen commitment can be extended to the Generalized Pedersen commitment that enables to commit to more than one message. To be more precise, the message space can be defined as $\mathcal{M} = \mathbb{Z}_p^n$, where n is an upper bound for the number of committed messages.

- $\mathsf{COM.Setup}(1^\lambda, n)$: Take security parameter, λ and an integer n as inputs. Sample $n+1$ random generators $\mathsf{G}, \mathsf{H}_1, \mathsf{H}_2, \ldots, \mathsf{H}_n \xleftarrow{\$} \mathbb{G}^{(n+1)}$. Return the public parameters $\mathsf{pp} = (\mathbb{G}, p, \mathsf{G}, \mathsf{H}_1, \ldots, \mathsf{H}_n)$ as output.
- $\mathsf{COM.Com}(\mathsf{pp}, \vec{m}, \tau)$: Take the public parameters pp, a message vector $\vec{m} := (m_1, \ldots, m_n)$ and random opening $\tau \in \mathbb{Z}_p$ as inputs. Output $\mathsf{cm} = \mathsf{G}^\tau \prod_{j=1} \mathsf{H}_i^{m_i}$.
- $\mathsf{COM.Verify}(\mathsf{pp}, \mathsf{cm}, \vec{m}', \tau')$: Compute $\mathsf{cm}' = \mathsf{G}^{\tau'} \prod_{j=1}^n \mathsf{H}_i^{m_i'}$. Return 1 if $\mathsf{cm} = \mathsf{cm}'$ and 0 otherwise.

Inner-product predicate encryption by Okamoto-Takashima We give a brief overview of what we briefly refer to as OT12 scheme [55]. While describing the full scheme is outside the scope of this overview section, we briefly describe the basics behind public-key generation, encryption and decryption. Assume we are in a bilinear group setting $\mathsf{pp} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \mathsf{G}_1, \mathsf{G}_2)$ as before. We describe there the predicate-only version already adapted to the asymmetric pairing case that we use in our implementation.

Public key and master secret. We first sample an invertible matrix X of dimension $N = 4n+2$ (where n is the number of attributes) with elements in $\mathbb{F}_p^*$ and consider the matrix $\psi \cdot X^{-1}$ for a random, non-zero field element ψ. For syntactical purposes only, the transpose is actually considered, i.e., we define $Y = \psi \cdot (X^{-1})^T$.

For notational purposes, we define basis vectors $\vec{a}_i = (\underbrace{1_{\mathbb{G}_2}, \ldots, 1_{\mathbb{G}_2}}_{i-1}, \mathsf{G}_2, \underbrace{1_{\mathbb{G}_2}, \ldots, 1_{\mathbb{G}_2}}_{N-i})$ and $\vec{a}^*_i = (\underbrace{1_{\mathbb{G}_1}, \ldots, 1_{\mathbb{G}_1}}_{i-1}, \mathsf{G}_1, \underbrace{1_{\mathbb{G}_1}, \ldots, 1_{\mathbb{G}_1}}_{N-i})$. For a an element $c \in \mathbb{F}_p$ the notation $c\vec{a}_i$ is shorthand for $(1_{\mathbb{G}_2}, \ldots, 1_{\mathbb{G}_2}, \mathsf{G}_2^c, 1_{\mathbb{G}_2}, \ldots, 1_{\mathbb{G}_2})$. And for two vectors $\vec{b} = (B_1, \ldots, B_N) \in \mathbb{G}_i$ and $\vec{b}' = (B'_1, \ldots, B'_N) \in \mathbb{G}_i$, we write $\vec{b} \odot \vec{b}' := (B_1 B'_1, \ldots, B_N B'_N)$. Finally, a pairing operation continued for vectors is defined: let $\vec{c} = (C_1, \ldots, C_N) \in \mathbb{G}_1^N$ and $\vec{c}' = (C'_1, \ldots, C'_N) \in \mathbb{G}_2^N$, then $\hat{e}(\vec{c}, \vec{c}') := \prod_{i=1}^N e(C_i, C'_i)$.

Armed with these tools and in particular the matrices $X = (x_{i,j})$ and $Y = (y_{i,j})$, we now define the public key and the master secret key: the public key $\mathbb{B} = (\vec{b}_1, \ldots, \vec{b}_N)$ consists of N vectors $\vec{b}_i := \bigodot_{j=1}^N x_{i,j}\vec{a}_j$. The master secret key $\mathbb{B}^* = (\vec{b}^*_1, \ldots, \vec{b}^*_N)$ consists of N vectors $\vec{b}^*_i := \bigodot_{j=1}^N y_{i,j}\vec{a}^*_j$.

We observe that there is the following relationship between $\mathbb{B}$ and $\mathbb{B}^*$ that follows from the definition of matrices X and Y:

$$\hat{e}(\vec{b}^*_i, \vec{b}_j) = e(\mathsf{G}_1, \mathsf{G}_2)^{x_{i,1}y_{j,1} + x_{i,2}y_{j,2} + \ldots + x_{i,N}y_{j,N}} = \begin{cases} e(\mathsf{G}_1, \mathsf{G}_2) =: \mathsf{G}_T, & \text{if } i = j, \\ 1_{\mathbb{G}_T}, & \text{if } i \neq j. \end{cases}$$

Key generation. For an attribute vector $\vec{v} \in \mathbb{F}_p^n \setminus \{\vec{0}\}$, one first samples $\sigma \leftarrow \mathbb{F}_p$ and $\vec{n} \leftarrow \mathbb{F}_p^n$ at random. We then form the vector $\vec{z}^* = (1, \sigma\vec{v}, \underbrace{0, \ldots, 0}_{2n}, \vec{n}, 0)$. The key for attribute $\vec{v}$ is defined as $\vec{k}^* := \bigodot_{i=1}^N z^*_i \vec{b}^*_i$.

Encryption. For encryption, which is done relative to attribute vector $\vec{x} \in \mathbb{F}_p^n \setminus \{\vec{0}\}$, one samples random values $\omega, \phi \leftarrow \mathbb{F}_p$ and defines the helper vector $\vec{z} := (1, \omega\vec{x}, \underbrace{0, \ldots, 0}_{3n}, \phi)$. The ciphertext is defined as $\vec{c} := \bigodot_{i=1}^N z_i \vec{b}_i$.

Decryption. In the predicate-only case, a key $\vec{k}^*$ decrypts a ciphertext $\vec{c}$ iff $\hat{e}(\vec{k}^*, \vec{c}) = \mathsf{G}_T$.

We observe, for correctness, that due to the above relation between $\mathbb{B}^*$ and $\mathbb{B}$, the operation $\hat{e}$ in fact computes the inner product of the vectors $\vec{z}^*$ and $\vec{z}$ in the exponent of G_T. That is, $\hat{e}(\vec{k}^*, \vec{c}) = e(\mathsf{G}_1, \mathsf{G}_2)^{1 + \omega\sigma\langle \vec{v}, \vec{x} \rangle}$, and therefore $\hat{e}(\vec{k}^*, \vec{c}) = \mathsf{G}_T$ when the inner product $\langle \vec{v}, \vec{x} \rangle$ is zero.

We refer to [55] for the proof that this scheme is attribute hiding in the sense defined in Section 3.7.

C.2 Proof Systems

In this section, we give some background on the proof systems we use in our implementation with a selection of useful basic protocols.

Sigma protocols. We first summarize the utilized sigma protocols (that is, the non-interactive versions via the Fiat-Shamir heuristic) in our efficient instantiation and give an overview on the techniques implied in their implementations. All the protocols are assumed a bilinear group setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \mathsf{G}_1, \mathsf{G}_2)$ and the Pedersen commitment.

Proving the Knowledge of Discrete Logarithm. Figure 22 describes a non-interactive sigma protocol that enables a prover to prove the knowledge of a scalar witness $a \in \mathbb{Z}_p$ under the public instance of $A = \mathsf{G}^a$, where G is a generator of a group $\mathbb{G}$ of prime order p. Note that the hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ is modeled in the random oracle model.

$\Sigma\text{-}\mathsf{Dlog}\{(a) \mid A = \mathsf{G}^a\}$
•$\mathsf{Prove}(\mathsf{CRS}, x, w)$: Takes the instance $x = (A)$ and the witness $w = (a)$ as inputs. It then samples $r \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $R = \mathsf{G}^r$ and challenge $c = H(A, R, \mathsf{G})$ and forms $z = r - ca \mod p$. It then returns the proof $\pi = (c, z, R)$ as output.
•$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$: Takes the instance $x = (A)$ and proof $\pi = (c, z, R)$ as inputs. It then computes $c' = H(A, R, \mathsf{G})$ and checks the equality of $c' = c$ and $R = A^c\mathsf{G}^z$. It returns 1 if they hold and 0 otherwise.

Fig. 22: Non-interactive proof of knowledge of Dlog.

Proving the Knowledge of a Committed value and its ElGamal encryption. Figure 23 describes a sigma protocol proving the knowledge of a palintext m encrypted based on ElGamal encryption and committed via Pedersen commitment. In other words, proving the knowledge of scalar message $m \in \mathbb{Z}_p$ such that $\mathsf{cm} = \mathsf{G}^m\mathsf{H}^e$ and simultanously we have, $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (\mathsf{G}_1^r, \mathsf{G}_1^m\mathsf{pk}^r)$. The hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ is modeled in the random oracle model.

Σ-$\mathsf{ElGamal}\{(m, r, e) \mid \mathsf{ct}_1 = \mathsf{G}_1^r \wedge \mathsf{ct}_2 = \mathsf{G}_1^m \mathsf{pk}^r \wedge \mathsf{cm} = \mathsf{G}^m \mathsf{H}^e\}$

•$\mathsf{Prove}(\mathsf{CRS}, x, w)$: Takes the instance $x = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{cm})$ and the witness $w = (m, r, e)$ as inputs. It then samples $r_1, r_2, r_3 \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $R_1 = \mathsf{G}_1^{r_1}$, $R_2 = \mathsf{G}_1^{r_2} \mathsf{pk}^{r_1}$ and $R_3 = \mathsf{COM.Com}(\mathsf{pp}, r_2; r_3) = \mathsf{G}^{r_2} \mathsf{H}^{r_3}$, the challenge $c = H(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{cm}, R_1, R_2, R_3, \mathsf{G}, \mathsf{H})$ and forms $z_1 = r_1 - cr \mod p$, $z_2 = r_2 - cm \mod p$ and $z_3 = r_3 - ce \mod p$. It then returns the proof $\pi = (c, z_1, z_2, z_3, R_1, R_2, R_3)$ as output.

•$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$: Takes the instance $x = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{cm})$ and proof $\pi = (c, z_1, z_2, z_3, R_1, R_2, R_3)$ as inputs. It then computes $c' = H(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{cm}, R_1, R_2, R_3, \mathsf{G}, \mathsf{H})$ and checks the equality of equations $c' = c$, $R_1 = \mathsf{ct}_1^c \mathsf{G}_1^{z_1}$, $R_2 = \mathsf{G}_1^{z_2} \mathsf{pk}^{z_1} \mathsf{ct}_2^c$ and $R_3 = \mathsf{G}^{z_2} \mathsf{H}^{z_3} \mathsf{cm}^c$. It returns 1 if all the equations hold; 0 otherwise.

Fig. 23: Non-interactive proof of knowledge of ElGamal encrypted value.

Proving the Equality of Committed Values in different groups. We extend the protocol proposed in [28] s.t. for a given cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p, Figure 24 describes a sigma protocol enabling a prover to prove that two commitments $\mathsf{cm}_1 = \mathsf{G}_1^m \mathsf{H}_1^{e_1} \mathsf{K}_1^{u_1}$ and $\mathsf{cm}_2 = \mathsf{G}_2^m \mathsf{H}_2^{e_2} \mathsf{K}_2^{u_2}$ are committing to the same message m. Note that $\mathsf{G}_1, \mathsf{H}_1, \mathsf{K}_1 \in \mathbb{G}_1$ and $\mathsf{G}_2, \mathsf{H}_2, \mathsf{K}_2 \in \mathbb{G}_2$ s.t. the discrete logarithms $\log_{\mathsf{G}_1}(\mathsf{H}_1)$, $\log_{\mathsf{G}_1}(\mathsf{K}_1)$ and $\log_{\mathsf{G}_2}(\mathsf{H}_2)$, $\log_{\mathsf{G}_2}(\mathsf{K}_2)$ are unknown to the prover. The hash function $H' : \{0,1\}^* \to \mathbb{Z}_{2^k}$, where k is a fixed integer and $2^k < p$ is modeled in the random oracle model.

Σ-$\mathsf{Bridging}\{(m, e_1, e_2, u_1, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^m \mathsf{H}_1^{e_1} \mathsf{K}_1^{u_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^m \mathsf{H}_2^{e_2} \mathsf{K}_2^{u_2}\}$

•$\mathsf{Prove}(\mathsf{CRS}, x, w)$: Takes the instance $x = (\mathsf{cm}_1, \mathsf{cm}_2)$ and the witness $w = (m, e_1, e_2, u_1, u_2)$ as inputs. It then samples $r_1, r_2, r_3, r_4, r_5 \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $R_1 = \mathsf{G}_1^{r_1} \mathsf{H}_1^{r_2} \mathsf{K}_1^{r_3}$, $R_2 = \mathsf{G}_2^{r_1} \mathsf{H}_2^{r_4} \mathsf{K}_2^{r_5}$ and the challenge $c = H'(\mathsf{cm}_1, \mathsf{cm}_2, R_1, R_2, \mathsf{G}_1, \mathsf{H}_1, \mathsf{K}_1, \mathsf{G}_2, \mathsf{H}_2, \mathsf{K}_2)$ and forms $z_1 = r_1 - cm \mod p$, $z_2 = r_2 - ce_1 \mod p$, $z_3 = r_3 - cu_1 \mod p$, $z_4 = r_4 - ce_2 \mod p$ and $z_5 = r_5 - cu_2 \mod p$. It then returns the proof $\pi = (c, z_1, z_2, z_3, z_4, z_5, R_1, R_2)$ as output.

•$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$: Takes the instance $x = (\mathsf{cm}_1, \mathsf{cm}_2)$ and proof $\pi = (c, z_1, z_2, z_3, z_4, z_5, R_1, R_2)$ as inputs. It then computes $c' = H'(\mathsf{cm}_1, \mathsf{cm}_2, R_1, R_2, \mathsf{G}_1, \mathsf{H}_1, \mathsf{K}_1, \mathsf{G}_2, \mathsf{H}_2, \mathsf{K}_2)$ and checks the equality of $c' = c$, $R_1 = \mathsf{cm}_1^c \mathsf{G}_1^{z_1} \mathsf{H}_1^{z_2} \mathsf{K}_1^{z_3}$ and $R_2 = \mathsf{cm}_2^c \mathsf{G}_2^{z_1} \mathsf{H}_2^{z_4} \mathsf{K}_2^{z_5}$. It returns 1 if the checks hold and 0 otherwise.

Fig. 24: Non-Interactive proof of Equality of Two Commitments.

Proving a Multiplicative Relation on Committed Values. Figure 25 describes a sigma protocol to prove the knowledge of committed values and also a multiplicative relation between them. More precisely, this protocol enables to prove the knowledge of integers x_1 and x_2 s.t. $x_3 = x_1x_2 \mod p$ by issuing the commitments $\mathsf{cm}_i = \mathsf{G}^{x_i}\mathsf{H}^{e_i}$ for $i = 1, 2, 3$. Note that the hash function $H' : \{0,1\}^* \to \mathbb{Z}_{2^k}$, where k is a fixed integer and $2^k < p$ is modeled in the random oracle model.

Σ-$\mathsf{MultCom}\{\{(x_j, e_j)\}_{j=1}^3 \mid \mathsf{cm}_i = \mathsf{G}^{x_i}\mathsf{H}^{e_i}$ for $i = 1, 2, 3 \ \wedge\ x_3 = x_1x_2 \mod p\}$
•$\mathsf{Prove}(\mathsf{CRS}, x, w)$: Takes the instance $x = (\mathsf{cm}_1, \mathsf{cm}_2, \mathsf{cm}_3)$ and the witness $w = (x_1, x_2, x_3, e_1, e_2, e_3)$ as inputs. It then samples $r_1, r_2, r_3 \xleftarrow{\$} \mathbb{Z}_p^*$ and $s, s_1, s_2, s_3 \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $R_i = \mathsf{G}^{r_i}\mathsf{H}^{s_i}$ and $R = \mathsf{cm}_1^{r_2}\mathsf{H}^s$, the challenge $c = H'(\mathsf{cm}_1, \mathsf{cm}_2, \mathsf{cm}_3, R, R_1, R_2, R_3, \mathsf{G}, \mathsf{H})$ and forms $z_i = r_i - cx_i \mod p$, $t_i = s_i - ce_i \mod p$ for $i = 1, 2, 3$ and $t = s - ce \mod p$, where $e = e_3 - e_1x_2 \mod p$. It then returns the proof $\pi = (c, z_1, z_1, z_3, t_1, t_2, t_3, t)$ as output.
•$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$: Takes the instance $x = (\mathsf{cm}_1, \mathsf{cm}_2, \mathsf{cm}_3)$ and proof $\pi = (c, z_1, z_1, z_3, t_1, t_2, t_3, t)$ as inputs. It then computes $c' = H'(\mathsf{cm}_1, \mathsf{cm}_2, \mathsf{cm}_3, R, R_1, R_2, R_3, \mathsf{G}, \mathsf{H})$ and checks the equality of equations $c' = c$, $R_i = \mathsf{G}^{z_i}\mathsf{H}^{t_i}\mathsf{cm}_i^c$ for $i = 1, 2, 3$ and $R = \mathsf{cm}_3^c\mathsf{cm}_1^{z_2}\mathsf{H}^t$. It returns 1 if they hold and 0 otherwise.

Fig. 25: Non-Interactive proof of multiplicative relation on committed values.

Proving the Knowledge of a DY PRF key and its Well-Formedness. Figure 26 recalls the DY PRF well-formedness protocol described in [28]. As part of this protocol, a prover using DY PRF key k shows that the PRF output is formed correctly under a given input ctr, i.e. $\mathsf{ID} = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr}) = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}$.

Σ-$\mathsf{PRF}\{(\mathsf{k}, \mathsf{ctr}) \mid \mathsf{ID} = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\}$
•$\mathsf{Prove}(\mathsf{CRS}, x, w)$: Takes the instance $x = (\mathsf{ID})$ and the witness $w = (\mathsf{k}, \mathsf{ctr})$ as inputs. Note that ID can be seen as a commitment of the form $\mathsf{COM.Com}(\mathsf{pp}, (1/\mathsf{k}+\mathsf{ctr}); 0) = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\mathsf{H}^0$. It samples $e_1, e_2, e_3 \xleftarrow{\$} \mathbb{Z}_p^*$ and computes the commitments $\mathsf{cm}_1 = \mathsf{COM.Com}(\mathsf{pp}, \mathsf{ctr}; e_1) = \mathsf{G}^{\mathsf{ctr}}\mathsf{H}^{e_1}$, $\mathsf{cm}_2 = \mathsf{COM.Com}(\mathsf{pp}, \mathsf{k}; e_2) = \mathsf{G}^{\mathsf{k}}\mathsf{H}^{e_2}$, $\mathsf{cm}_3 = \mathsf{COM.Com}(\mathsf{pp}, (\mathsf{k}+\mathsf{ctr}); e_3) = \mathsf{G}^{(\mathsf{k}+\mathsf{ctr})}\mathsf{H}^{e_3}$. It then runs $\pi \leftarrow \Sigma$-$\mathsf{MultCom.Prove}(\mathsf{CRS}, x_1, w_1)$ with input commitments $x_1 = (\mathsf{cm}_1 \cdot \mathsf{cm}_2, \mathsf{ID}, \mathsf{G})$ and $w_1 = (\mathsf{k} + \mathsf{ctr}, 1/(\mathsf{k} + \mathsf{ctr}), 1)$. It returns the proof π as output.
•$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$: Takes the instance $x = (\mathsf{cm}_1 \cdot \mathsf{cm}_2, \mathsf{ID}, \mathsf{G})$ and proof π as inputs. It then checks the validity of the proof by running Σ-$\mathsf{MultCom.Verify}(\mathsf{CRS}, x, \pi)$.

Fig. 26: Non-interactive proof of knowledge of DY's PRF key and its well-formedness.

Proving the Knowledge of opening of Generalized Pedersen Commitment. Figure 27 recalls the proving knowledge of opening in a Generalized Pedersen commitment protocol described in [44]. As part of this protocol, a prover using a vector of messages $\vec{m}$ shows that the commitment cm is computated correctly and it has the knowledge of opening τ under a given public parameter pp, i.e. $\mathsf{cm} = \mathsf{COM.Com}(\mathsf{pp}, \vec{m}, \tau) = \mathsf{G}^{\tau} \prod_{i=1}^{n} \mathsf{H}_i^{m_i}$.

$\Sigma\text{-}\mathsf{GPedCom}\{(m_1, \ldots, m_n, \tau) \mid \mathsf{cm} = \mathsf{G}^{\tau} \prod_{i=1}^{n} \mathsf{H}_i^{m_i}\}$
•$\mathsf{Prove}(\mathsf{CRS}, x, w)$: Takes the instance $x = (\mathsf{cm}, \mathsf{G}, \mathsf{H}_1, \ldots, \mathsf{H}_n)$ and the witness $w = (m_1, \ldots, m_n, \tau)$ as inputs. It samples $\vec{x} = (x_1, \ldots, x_n) \xleftarrow{\$} \mathbb{Z}_p^n$ and $\tau_x \leftarrow \mathbb{Z}_p$ and computes $\mathsf{cm}_x = \mathsf{G}^{\tau_x} \prod_{i=1}^{n} \mathsf{H}_i^{x_i}$. It then computes the challenge $c = H'(\mathsf{cm}, \mathsf{cm}_x, \mathsf{G}, \mathsf{H}_1, \ldots, \mathsf{H}_n)$ and forms $z_i = x_i + c \cdot m_i \mod p$ for $i \in [1, n]$ and $t = \tau_x + c \cdot \tau \mod p$. It returns the proof $\pi = (\mathsf{cm}_0, z_1, \ldots, z_n, t)$ as output.   •$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$: Takes the instance $x = (\mathsf{cm}, \mathsf{G}, \mathsf{H}_1, \ldots, \mathsf{H}_n)$ and proof $\pi = (\mathsf{cm}_0, z_1, \ldots, z_n, t)$ as inputs. It then computes $c' = H'(\mathsf{cm}, \mathsf{cm}_x, \mathsf{G}, \mathsf{H}_1, \ldots, \mathsf{H}_n)$ and checks the equality of equations $c' = c$, $\mathsf{cm}_x \cdot \mathsf{cm}^c = \mathsf{COM.Com}(\vec{z}, t) = \mathsf{G}^t \prod_{i=1}^{n} \mathsf{H}_i^{z_i}$. It returns 1 if they hold and 0 otherwise.

Fig. 27: Non-Interactive proof of knowledge of opening of Generalized Pedersen commitments.

Groth-Sahai proofs. GS proofs [45] are able to prove the satisfiability of some quadratic equations in bilinear setting. However, in this paper, we only use GS proofs to demonstrate pairing product equations satisfiability of the following form,

$$\prod_{i=1}^{n} e(A_i, \mathcal{Y}_i) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{j=1}^{m} \prod_{i=1}^{n} e(\mathcal{X}_j, \mathcal{Y}_i)^{\gamma_{i,j}} = T \ ,$$

where $\mathcal{X}_1, \ldots, \mathcal{X}_m \in \mathbb{G}_1$, $\mathcal{Y}_1, \ldots, \mathcal{Y}_n \in \mathbb{G}_2$ are the witnesses given as a commitment and $T \in \mathbb{G}_T$, $A_1, \ldots, A_n \in \mathbb{G}_1$, $B_1, \ldots, B_m \in \mathbb{G}_2$ and $\Gamma := \{\gamma_{i,j}\}_{i \in [1,m], j \in [1,n]} \in \mathbb{Z}_p^{m \times n}$.

GS proofs are essentially commit-and-prove systems, in which the prover proves that a quadratic equation satisfies using the committed assignments. Therefore, there are two steps: first, the prover commits to the values, and then it proves their validity through some relation. This scheme can be instantiate in two possible settings: non-interactive witness-indistinguishable (NIWI) and non-interactive zero-knowledge (NIZK). If in the described PPE, the constant value $T = 1_{\mathbb{G}_T}$ this construction can guarantee Zero-Knowledge property[10]. Thus in the rest of this section we take this condition into account. In the following, we briefly summarize the most efficient instantiation of GS proofs based on SXDH assump-

[10] According to Escala and Groth [32], the proof system remains zero-knowledge if the base element for the group and public constant are paired to each other.

tion (i.e. DDH holds in both source groups $\mathbb{G}_1$ and $\mathbb{G}_2$), both in terms of proof size and number of basic pairings required in the verification phase.[11]

Extended bilinear maps, $E : \mathbb{G}_1^2 \times \mathbb{G}_2^2 \to \mathbb{G}_T^4$, are a generalization for the standard bilinear pairings, defined in Definition 1. For any given group elements $a_1, a_2 \in \mathbb{G}_1$ and $b_1, b_2 \in \mathbb{G}_2$, an extended bilinear map (tensor product) is defined as follows:

$$E\left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_1 \; b_2 \end{pmatrix}\right) = \begin{pmatrix} e(a_1, b_1) \; e(a_1, b_2) \\ e(a_2, b_1) \; e(a_2, b_2) \end{pmatrix} \; .$$

GS proofs additionally rely on a variation of the Pedersen commitments, which are discussed in Appendix C.1, where the commitments are generated based on two generators rather than a single one. Loosely speacking, this special commitment scheme enables proof simulation. The double generator Pedersen commitment over a cyclic group $\mathbb{G} = \langle \mathsf{G} \rangle$ with a prime order p consists of the following PPT algorithms:

– $\mathsf{pp} \leftarrow \mathsf{COM.Setup}(1^\lambda)$: Take security parameter, λ in its unary representation as input. Sample $\mathsf{H}_1, \mathsf{H}_2 \xleftarrow{\$} \mathbb{G}$. Return the public parameters $\mathsf{pp} = (\mathbb{G}, p, \mathsf{G}, \mathsf{H}_1, \mathsf{H}_2)$ as output.
– $\mathsf{cm} \leftarrow \mathsf{COM.Com}(\mathsf{pp}, M; \tau_1, \tau_2)$: Take public parameters pp, a message $M \in \mathbb{G}$ and random openings τ_1, τ_2 as inputs. Output $\mathsf{cm} = M\mathsf{H}_1^{\tau_1}\mathsf{H}_2^{\tau_2}$.
– $0/1 \leftarrow \mathsf{COM.Verify}(\mathsf{pp}, \mathsf{cm}, M', \tau_1', \tau_2')$: Compute $\mathsf{cm}' = M'\mathsf{H}_1^{\tau_1'}\mathsf{H}_2^{\tau_2'}$. Return 1, if $\mathsf{cm} = \mathsf{cm}'$; otherwise return 0.

Next we outline GS proofs for a simple case based on SXDH assumption that the prover aims to prove the knowledge of group elements $\mathcal{X}, \mathcal{Y} \in \mathbb{G}_1 \times \mathbb{G}_2$ as witnesses s.t. $e(\mathcal{X}, \mathcal{Y})^\gamma = T$, where $T \in \mathbb{G}_T$ is a known constant value.

– $\mathsf{CRS} \leftarrow \mathsf{GS.Setup}(1^\lambda)$: Take the security parameter λ as input. Sample eight group elements $\mathsf{CRS} := (\mathsf{H}_1, \mathsf{H}_2, \mathsf{K}_1, \mathsf{K}_2, \mathsf{U}_1, \mathsf{U}_2, \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathbb{G}_1^4 \times \mathbb{G}_2^4$. It then returns CRS as output.
– $\pi \leftarrow \mathsf{GS.Prove}(\mathsf{CRS}, x, w)$. It takes CRS, witness $w = (\mathcal{X}, \mathcal{Y}) \in \mathbb{G}_1 \times \mathbb{G}_2$ and instance $x = (T)$ as inputs. It samples the random integers $r_1, r_2, s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p^*$ and commits to witness by computing $(a_1, a_2) = (\mathsf{H}_1^{r_1}\mathsf{H}_2^{r_2}, \mathcal{X}\mathsf{K}_1^{r_1}\mathsf{K}_2^{r_2})$ and $(b_1, b_2) = (\mathsf{U}_1^{s_1}\mathsf{U}_2^{s_2}, \mathcal{Y}\mathsf{V}_1^{s_1}\mathsf{V}_2^{s_2})$. It samples the random integers $\alpha, \beta, \zeta, \delta \xleftarrow{\$} \mathbb{Z}_p$ and then generates the proofs $\phi_1 = (b_1^{\gamma r_1}\mathsf{U}_1^{\alpha}\mathsf{V}_1^{\beta} \;\; b_2^{\gamma r_2}\mathsf{U}_2^{\alpha}\mathsf{V}_2^{\beta})$, $\phi_2 = (b_1^{\gamma s_1}\mathsf{U}_1^{\zeta}\mathsf{V}_1^{\delta} \;\; b_2^{\gamma s_2}\mathsf{U}_2^{\zeta}\mathsf{V}_2^{\delta})$, $\theta_1 = (\mathsf{H}_1^{-\alpha}\mathsf{K}_1^{-\zeta} \;\; \mathcal{X}^{r_2}\mathsf{H}_2^{-\alpha}\mathsf{K}_2^{-\zeta})$ and $\theta_2 = (\mathsf{H}_1^{-\beta}\mathsf{K}_1^{-\delta} \;\; \mathcal{X}^{\gamma s_2}\mathsf{H}_2^{-\beta}\mathsf{K}_2^{-\delta})$, and return the proof $\pi = (a_1, a_2, b_1, b_2, \phi_1, \phi_2, \theta_1, \theta_2)$ as output.
– $0/1 \leftarrow \mathsf{GS.Verify}(\mathsf{CRS}, x, \pi)$: It takes CRS, the instance x and proof π as inputs. It then checks the validity of the following pairing product equation:

$$E\left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} b_1 \; b_2 \end{pmatrix}^\gamma\right) =$$
$$E\left(\begin{pmatrix} \mathsf{H}_1 \\ \mathsf{H}_2 \end{pmatrix}, \phi_1\right) E\left(\begin{pmatrix} \mathsf{K}_1 \\ \mathsf{K}_2 \end{pmatrix}, \phi_2\right) E\left(\theta_1, \begin{pmatrix} \mathsf{U}_1 \; \mathsf{U}_2 \end{pmatrix}\right) E\left(\theta_2, \begin{pmatrix} \mathsf{V}_1 \; \mathsf{V}_2 \end{pmatrix}\right) \begin{pmatrix} T & 1_{\mathbb{G}_T} \\ 1_{\mathbb{G}_T} & 1_{\mathbb{G}_T} \end{pmatrix} \; .$$

If the equation holds it returns 1 and accepts the proof; 0 otherwise.

[11] To implement the Groth-Sahai proofs, we modified the GS implementation provided in this repository.

This simple example results in a single $\gamma \in \{-1, 0, 1\}$ value because $m = n = 1$. In general, however, Γ is a matrix of dimension $m \times n$. Throughout the next section, we discuss how this matrix can be defined for different PPEs.

Herold et al.'s batching technique [47]. To check the validity of a GS proof for any PPE consisting of n first-group elements and m second-group elements, a verifier must compute $4(n + m + 4)$ pairings. However, Herold et al. described a batching technique in [47] that reduces the number of pairings by a factor of 4. This means that a verifier checking the same proof needs to compute only $n + m + 4$ pairings, which is a significant improvement, especially in real-world use cases. The authors replace an extended pairing product equation to a basic pairing product equation using linear algebra. As a simple example, for a given group vectors $\vec{a} \in \mathbb{G}_1^2$ and $\vec{b} \in \mathbb{G}_2^2$ the extended bilinear equation can be written as follows:

$$E\left(\vec{a}, \vec{b}\right) = \begin{pmatrix} e(a_1, b_1) \; e(a_1, b_2) \\ e(a_2, b_1) \; e(a_2, b_2) \end{pmatrix} = \begin{pmatrix} t_1 \; t_2 \\ t_3 \; t_4 \end{pmatrix} \ ,$$

where $t_i \in \mathbb{G}_T$ for $i \in [1, 4]$. A verifier computes $A = a_1^{r_1} a_2^{r_2}$ and $B = b_1^{s_2} b_2^{s_2}$ using the randomnesses $r_1, r_2, s_1, s_2 \in \mathbb{Z}_p^*$. In this case, the above extended pairing product equation can be rewritten as follows:

$$e(A, B) = t_1^{r_1 s_1} t_2^{r_1 s_2} t_3^{r_2 s_1} t_4^{r_2 s_2} \ .$$

It is easy to see that the above technique is correct. However, the soundness error is at most $2/p$. More interestingly, it reduces the number of pairings by 75% compared to the naive approach. Specifically, it only requires a single pairing instead of 4 pairings.

Range-proofs. Range-proofs enable a prover to prove a committed value x computed as $\mathsf{cm} = \mathsf{COM.Com}(\mathsf{pp}, x, r)$ is in the range of $[0, 2^n)$.[12]

D Realization of NIZK relations

Next, we give a detailed description of the languages in the proposed ul-PCS constructions and the used techniques for their implementation. Note that for the ease of following we use the *gray* background to highlight the hidden values that should be considered as witnesses in each relation. Additionally, the described relations are given solely on their own, while the prover is expected to make a bridge between them. As the underlying proof systems rely on the commit-and-prove principle, and a commitment to a witness is issued by the prover, we can bridge the relations by applying the sigma protocol described in Figure 24 whenever a hidden parameter is used in more than one relation. In what follows, a bridging proof is indicated by $\blacklozenge$. As it is illustrated in Figure 17, the realtions are proved with three main proof systems, including $\boxed{\text{sigma protocols}}$, range-proofs and Groth-Sahai (GS) proofs

[12] To implement the range-proof, we use the open-source bulletproof Python implementation available in this repository.

D.1 Generic ul-PCS instantiated with Inner-Product Predicate Encryption

Language $\mathcal{L}_1$. The first language in the generic ul-PCS takes the instances $x_{\mathsf{st}} = (T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{A}, \mathsf{mpk}_{\mathsf{PE}})$ and the witness $w_{\mathsf{st}} := (\mathsf{k}, \mathsf{ctr}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, x, \sigma_{\mathsf{sig}}^{1}, \sigma_{\mathsf{sig}}^{2}, \sigma_{\mathsf{ctr}})$ as inputs and the prover proves the satisfiability of the following relations:

$\mathcal{L}_1$.**1**. $\boxed{\mathsf{ID}_{\mathsf{ctr}} = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})}$: As discussed on Appendix C.1, we use the DY PRF schemes. Thus we use the sigma protocol described in Figure 26 to prove its well-formedness, i.e. $\Sigma\text{-}\mathsf{PRF}\{(\mathsf{k}, \mathsf{ctr}) \mid \mathsf{ID}_{\mathsf{ctr}} = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\}$.

$\mathcal{L}_1$.**2**. $\mathsf{ctr} < T_{\mathsf{Rand}}$: Additionally, the prover utilizes the range-proof techniques to prove $\mathsf{ctr} \in [0, T_{\mathsf{Rand}})$.

$\blacklozenge$ To show that the used ctr in the above proofs are the same, the prover use the bridging sigma protocols described in Figure 24. More precisely, it runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{ctr}, e_1, e_2, 0, 0) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{ctr}}\mathsf{H}_1^{e_1} \;\wedge\; \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{ctr}}\mathsf{H}_2^{e_2}\}$, where cm_1 is obtained via Σ-PRF protocol while cm_2 is computed by the range-proof protocol. Note that the generators $\mathsf{G}_1, \mathsf{H}_1, \mathsf{G}_2, \mathsf{H}_2$ are random elements of any cyclic group.

$\mathcal{L}_1$.**3**. $\boxed{\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A}, (\mathsf{k}, x), \sigma_{\mathsf{sig}}^{1}) = 1}$: We instantiate this signature with the recalled SPS scheme in Appendix C.1. To prove the knowledge of a valid SPS signature $\sigma_{\mathsf{sig}}^{1}$ on hidden message $\vec{M} = (\mathsf{G}_1^{\mathsf{k}}, \mathsf{G}_1^{x})$ that is signed by the CA can be written as a PPE of the form, $e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^{A}\right) e\left(\mathsf{G}_1^{x}, \hat{X}_2^{A}\right) = e\left(R_{\mathsf{sig}}^{1}, T_{\mathsf{sig}}^{1}\right) \;\wedge\; e(S_{\mathsf{sig}}^{1}, \mathsf{G}_2) = e(\mathsf{G}_1, T_{\mathsf{sig}}^{1})$, where $\mathsf{vk}_{\mathsf{sig}}^{A} := (\hat{X}_1^{A}, \hat{X}_2^{A})$ and $\sigma_{\mathsf{sig}}^{1} := (R_{\mathsf{sig}}^{1}, S_{\mathsf{sig}}^{1}, T_{\mathsf{sig}}^{1})$. We use GS proof systems to show the satisfiability of this equation.

$\blacklozenge$ To demonstrate that the used k in the first relation and the obove relation are the same, the prover use the bridging sigma protocols described in Figure 24. More precisely, it runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \;\wedge\; \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$, where cm_1 is obtained via Σ-PRF protocol while cm_2 is computed by the GS proof systems.

$\mathcal{L}_1$.**4**. $\boxed{\mathsf{ct}_{\mathsf{ctr}} = \mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, x)}$: To prove the well-formedness of the ciphertext $\mathsf{ct}_{\mathsf{ctr}}$ obtained from $\mathsf{PE.Enc}(\mathsf{mpk}_{\mathsf{PE}}, x)$ algorithm in the key re-randomization phase and demonstrate the fact that the attributes x are certified by the CA and folded with the PRF seed k, i.e. $\mathsf{SPS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A}, (\mathsf{G}_1^{\mathsf{k}}, \mathsf{G}_1^{x}), \sigma_{\mathsf{sig}}^{1}) = 1$, we must make a few observations. Recall that the statements together should assure that the ciphertext is a correct encryption of some attribute x, and that this attribute is linked to the particular party's actual seed k (which it uses to provable derive its public pseudo-random identifier). The trick to obtain an implementation of this is fourfold:

(a) We employ OT12 as our POPE scheme described in Appendix C.1 and observe that the computation of the ciphertext $\vec{c} := \bigodot_{i=1}^{N} z_i \vec{b}_i$, which means in a component-wise notation that $c_i = \vec{b}_1[i]_1^{z} \cdot \vec{b}_2[i] \cdot \cdots \cdot \vec{b}_N[i]^{z_N}$, for $\vec{z} := (1, x_1, \ldots, x_n, \underbrace{0, \ldots, 0}_{3n}, \phi)$. This computation is very close to a generalized Pedersen commitment w.r.t. the vector of generators $(\vec{b}_1[i], \ldots, \vec{b}_N[i])$.

(b) Generalized Pedersen Commitments have a homomorphic property, and thus it is easy to, for a commitment cm to some vector $\vec{x}$, a commitment to $\omega\vec{x}$ by raising the commitment to the power of ω.

(c) The next step is to connect it to SPS-EQ in order to transfer the issuance of attributes by the authority to re-randomizations following the idea of Section 5.3 (cf. also Appendix D.3). This means that if the authority issues an initial OT12 ciphertext, in the form of an N generalized Pedersen commitments $\mathsf{cm}_{\vec{x},i}$ that are blindings of the c_i values above for the vector $\vec{z} := (1, x_1, \ldots, x_n, \underbrace{0, \ldots, 0}_{3n}, 0)$, together with an SPS-EQ signature on that vector, then a party is able to generate, using the homomorphic property, a commitment to a scaled vector on its attributes $\mathsf{cm}_{\omega\vec{x},i}$ (as required by OT12), and by the signature adaptation and unforgeability property of the SPS-EQ scheme it can indeed be verified that this is done correctly. By using the same trick as in Section 5.3, we can further bind this vector specifically to a party (see next point).
To randomize the ciphertext correctly according to OT12, we further compute a commitment to a vector $\vec{\phi_i} = (1, \underbrace{0, \ldots, 0}_{n}, \phi_i, \underbrace{0, \ldots, 0}_{3n-1}, \phi')$, where $\phi_i, \phi_2 \xleftarrow{\$} \mathbb{Z}_p^*$ and $n = [\frac{N-2}{4}]$, and prove knowledge of the opening (cf. Figure 27), in particular, this includes that we verify that the zero-positions are indeed zero (or alternatively, that is indeed a vector of length 3) [44]. The Generalized Pedersen commitments of this vector under the same basis denoted by $\vec{\mathsf{cm}}_\phi = (\mathsf{cm}_{\phi_1}, \ldots, \mathsf{cm}_{\phi_N})$ and can be homomorphically combined with the N commitments $\mathsf{cm}_{\omega\vec{x},i}$ to yield N commitments cm_i^{OT12} to the OT12 ciphertext components ct_i, where each component is now encoding the vector $\vec{z_i} := (1, x_1, \ldots, x_n, \phi_i, \underbrace{0, \ldots, 0}_{3n-1}, \phi')$. We reveal ct_i by revealing the final randomness of the Pedersen commitment. We further need to prove knowledge of the committed vector (using Figure 27 where the verifier can use the revealed r_i directly) in order to be formally extractable.[13] Note that some care must be taken when revealing the r_i, which is why we enocde vectors $\vec{z_i} := (1, x_1, \ldots, x_n, \phi_i, \underbrace{0, \ldots, 0}_{3n-1}, \phi')$ and not just $(1, x_1, \ldots, x_n, \underbrace{0, \ldots, 0}_{3n}, \phi')$. The additional randomness contribution, ϕ_i injected at a position that does not affect the inner-product computation of OT12 ensures that we can reveal r_i (masking the ith component) without leaking information about the intermediate computations relevant for OT12 in the commitments cm_{ϕ_i} and $\mathsf{cm}_{\omega\vec{x},i}$, in particular, this can be thought of as masking ϕ'.
Now, all ingredients are in place: a verifier is able to retrace the computation (where we put all elements required to do so in the proof string), and verify the SPS-EQ signature to be sure the OT12 ciphertext is correctly formed and connected to the attribute that was issued to the party.

(d) Finally, in order to link the party's seed k to this vector, we apply the same trick as in Section 5.3 (cf. also Appendix D.3) and create an accumulator A_k to which we

[13] This step could be omitted in an implementation to improve efficiency while trading provable for heuristic security. This appears acceptable in environments where a CRS is established using a ceremony to ensure that no trapdoor does exist in the system.

add k (cf. Section 3.5). We add the pair $(\mathsf{A_k}, \mathsf{G}_2, \dots)$ to the above vector. As shown in Section 5.3 and Appendix D.3, re-randomizing both elements $(\mathsf{A_k}, \mathsf{G}_2)$ preserves the relationship to prove that an element, in this case k, is in the accumulator. Thus, the party cannot only present a randomized vector (which is a commitment to the scaled attributes $\vec{x}$ as required by OT12), but also that this vector has been issued in connection with the seed k (which it uses to develop the PRF as described below) by proving that it has the corresponding accumulator witness. This completes the high-level realization of the two assertions above.

$\boxed{\blacklozenge}$ To demonstrate that the used $\mathsf{A_k}$ in the vector $(\mathsf{A_k}, \dots)$ is a valid accumulator value under the same PRF seed k in the first relation we first run a GS proof to prove the accumulator verification holds under k. Additionally we use the bridging sigma protocols described in Figure 24 to show this seed is the same as the one in the first relation on the well-formedness of PRF. More precisely, it runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$, where cm_1 is obtained via Σ-PRF protocol while cm_2 is computed by the GS proof systems on the validity of the accumulator verification.

$\mathcal{L}_1$**.5**. $\boxed{\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}), \sigma_{\mathsf{sig}}^{2}) = 1}$: As we discuss in Appendix C.1, this signature scheme is instantiated by a SPS. The knowledge of a SPS signature σ_{sig}^2 on hidden message $\vec{M} = (\mathsf{G}_1^{\mathsf{k}}, \mathsf{vk}_{\mathsf{sig}})$ that is signed by the CA can be written as a PPE of the form, $e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^A\right) e\left(\mathsf{vk}_{\mathsf{sig}}, \hat{X}_2^A\right) = e\left(R_{\mathsf{sig}}^2, T_{\mathsf{sig}}^2\right) \wedge e(S_{\mathsf{sig}}^2, \mathsf{G}_2) = e(\mathsf{G}_1, T_{\mathsf{sig}}^2)$, where $\mathsf{vk}_{\mathsf{sig}}^A := (\hat{X}_1^A, \hat{X}_2^A)$ and $\sigma_{\mathsf{sig}}^2 := (R_{\mathsf{sig}}^2, S_{\mathsf{sig}}^2, T_{\mathsf{sig}}^2)$. We use GS proof systems to show the satisfiability of this equation.

$\boxed{\blacklozenge}$ The prover additionally runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$, where cm_1 is obtained via Σ-PRF protocol while cm_2 is computed by the GS proof systems on the knowledge of SPS signature σ_{sig}^2.

$\mathcal{L}_1$**.6**. $\boxed{\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}, (\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ID}_{\mathsf{ctr}}), \sigma_{\mathsf{ctr}}) = 1}$: This signature is instantiated by the BLS signature, discussed in Appendix C.1. The prover should prove the satisfiability of the PPE relation described below in order to validate a newly generated verification key, $\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}$, and to bind it with the new identifier $\mathsf{ID}_{\mathsf{ctr}}$, $e\left(\mathsf{vk}_{\mathsf{sig}}, H(\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}} || \mathsf{ID}_{\mathsf{ctr}})\right) = e(\mathsf{G}_1, \sigma_{\mathsf{ctr}})$ that represents the validity of BLS signature. We use GS proof systems to instantiate this relation in zero-knowledge.

$\boxed{\blacklozenge}$ To show that the $\mathsf{vk}_{\mathsf{sig}}$ element in the relations discussed in $\mathcal{L}_1$.5 and $\mathcal{L}_1$.6 are identical we need to make a bridge between them. Due to the fact that both of these relations are proven via GS proof systems, we can instead combine them as follows [45]:

$$
\begin{aligned}
& e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^A\right)^1 e\left(\mathsf{vk}_{\mathsf{sig}}, \hat{X}_2^A\right)^1 e\left(R_{\mathsf{sig}}^2, T_{\mathsf{sig}}^2\right)^{-1} = 1_{\mathbb{G}_T} \wedge \\
& e(S_{\mathsf{sig}}^2, \mathsf{G}_2)^1 e(\mathsf{G}_1, T_{\mathsf{sig}}^2)^{-1} = 1_{\mathbb{G}_T} \wedge \\
& e\left(\mathsf{vk}_{\mathsf{sig}}, H(\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}} || \mathsf{ID}_{\mathsf{ctr}})\right)^1 e(\mathsf{G}_1, \sigma_{\mathsf{ctr}})^{-1} = 1_{\mathbb{G}_T} .
\end{aligned}
$$

This PPE involves both relations in $\mathcal{L}_1.5$ and $\mathcal{L}_1.6$ and we make sure to use the same commitment to the group element $\mathsf{vk}_{\mathsf{sig}}$ [45].

Language $\mathcal{L}_2$. In the second relation of the proposed generic ul-PCS scheme, the prover takes the statement $x_{\mathsf{st}} := (\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}^A_{\mathsf{sig}})$ and the witness $w_{\mathsf{st}} := (\mathsf{k}, \mathsf{ctr}, \mathsf{sk}_{f_x}, \sigma^2_{\mathsf{sig}})$ as inputs and acts as follows:

$\mathcal{L}_2$**.1**. $\boxed{\mathsf{ID}_S = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})}$: We use the sigma protocols to demonstrate the well-formedness of DY PRF, by running $\Sigma\text{-}\mathsf{PRF}\{(\mathsf{k}, \mathsf{ctr}) \mid \mathsf{ID}_S = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\}$, as described in Figure 26.

$\mathcal{L}_2$**.2**. $\mathsf{PE.Dec}(\mathsf{sk}_{f_x}, \mathsf{ct}_R) = 1$: To prove the knowledge of an PE secret key $\mathsf{sk}_{f_x} := (\mathsf{sk}_1, \dots, \mathsf{sk}_N)$ and proving the fact that it correctly decrypts the receiver's PE ciphertext $\mathsf{ct}_R := (\mathsf{ct}_1, \dots, \mathsf{ct}_N)$ to the identity value $1_{\mathbb{G}_T}$, we utilize the GS proof systems. As we already discussed in Appendix C.1 the OT12's decryption algorithm can be formalized with a PPE equation of the form, $\prod_{j=1}^N e\left(\mathsf{sk}_j, \mathsf{ct}_j\right) = e(\mathsf{G}_1, \mathsf{G}_2)$.

$\mathcal{L}_2$**.3**. $\mathsf{DS.Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{sk}_{f_x}), \sigma^3_{\mathsf{sig}}) = 1$: This signature is also instantiated by a SPS and similar to the previous languages, to prove the knowledge of a SPS signature σ^3_{sig} on hidden message $\vec{M} = (\mathsf{G}_1^{\mathsf{k}}, \mathsf{sk}_{f_x})$ that is signed by the CA we can use the GS proof systems. Towards the arithmetization of this relation we can write the verification equation with a PPE of the form, $e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}^A_0\right)\left(\prod_{j=1}^N e\left(\mathsf{sk}_j, \hat{X}^A_j\right)\right) = e\left(R^3_{\mathsf{sig}}, T^3_{\mathsf{sig}}\right) \wedge e(S^3_{\mathsf{sig}}, \mathsf{G}_2) = e(\mathsf{G}_1, T^3_{\mathsf{sig}})$, where $\mathsf{sk}_{f_x} := (\mathsf{sk}_1, \dots, \mathsf{sk}_N)$, $\mathsf{vk}^A_{\mathsf{sig}} := (\hat{X}^A_0, \hat{X}^A_1, \dots, \hat{X}^A_N)$ and $\sigma^3_{\mathsf{sig}} := (R^3_{\mathsf{sig}}, S^3_{\mathsf{sig}}, T^3_{\mathsf{sig}})$.

◆ Additionally, the prover runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$, where cm_1 is obtained via $\Sigma\text{-}\mathsf{PRF}$ protocol while the commitment cm_2 is computed by the GS proof systems on the knowledge of SPS signature σ^3_{sig}.

$\boxed{\blacklozenge}$ To demonstrate the fact that the used PE's secret key sk_{f_x} in the relations discussed in $\mathcal{L}_2.2$ and $\mathcal{L}_2.3$ are the same, the prover makes a bridge between them. Due to the fact that both of these relations are proven via GS proof systems, we can instead combine them and prove the following PPE.

$$
\begin{aligned}
&e(\mathsf{G}_1, \mathsf{G}_2)^{-1} \prod_{j=1}^{N} e\left(\mathsf{sk}_j, \mathsf{ct}_j\right)^1 = 1_{\mathbb{G}_T} \wedge \\
&e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}^A_0\right)^1 \prod_{j=1}^{N} e\left(\mathsf{sk}_j, \hat{X}^A_j\right)^1 e\left(R^3_{\mathsf{sig}}, T^3_{\mathsf{sig}}\right)^{-1} = 1_{\mathbb{G}_T} \wedge \\
&e(S^3_{\mathsf{sig}}, \mathsf{G}_2)^1 e(\mathsf{G}_1, T^3_{\mathsf{sig}})^{-1} = 1_{\mathbb{G}_T} \cdot
\end{aligned}
$$

This PPE involves both relations in $\mathcal{L}_2.2$ and $\mathcal{L}_2.3$ using the same commitment to group element sk_{f_x}.

D.2 ul-PCS for Separable Policies

Language $\mathcal{L}_1$. The first language in the ul-PCS with separable policies takes the instance $x_{\mathsf{st}} := \left(T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{vk}_{\mathsf{sig}}^{A,R}, \mathsf{pk}_{\mathsf{PKE}}^{A}\right)$ and witness $w_{\mathsf{st}} := \left(\mathsf{k}, \mathsf{ctr}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, m_x, \sigma_{\mathsf{sig}}^1, \sigma_{\mathsf{ctr}}\right)$ as inputs and then the prover proves the satisfiability of the following relations:

$\mathcal{L}_1$**.1.** $\boxed{\mathsf{ID}_{\mathsf{ctr}} = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})}$: To prove the well-formedness of the DY PRF, we use the sigma protocol described in Figure 26 and the prover runs $\Sigma\text{-}\mathsf{PRF}\left\{(\mathsf{k}, \mathsf{ctr}) \mid \mathsf{ID}_{\mathsf{ctr}} = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\right\}$.

$\mathcal{L}_1$**.2.** $\mathsf{ctr} < T_{\mathsf{Rand}}$: Additionally to prove $\mathsf{ctr} \in [0, T_{\mathsf{Rand}})$, the prover uses the range-proofs.

$\blacklozenge$ To demonstrate the fact that the used ctr in the above proofs are identical, the prover utilizes the bridging sigma protocol described in Figure 24. More precisely, it runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{ctr}, e_1, e_2, 0, 0) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{ctr}}\mathsf{H}_1^{e_1} \ \wedge\ \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{ctr}}\mathsf{H}_2^{e_2}\}$, where cm_1 is obtained via Σ-PRF protocol while the commtiment cm_2 is computed by the range-proof protocol.

$\mathcal{L}_1$**.3.** $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}^{A,R}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}, m_x), \sigma_{\mathsf{sig}}^1) = 1$: To prove the knowledge of a valid SPS signature σ_{sig}^1 on message $\vec{M} = (\mathsf{G}_1^{\mathsf{k}}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{G}_1^{m_x})$ signed by the CA we can prove the satisfiability of the PPE of the form, $e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^{A,R}\right) e\left(\mathsf{vk}_{\mathsf{sig}}, \hat{X}_2^{A,R}\right) e\left(\mathsf{G}_1^{m_x}, \hat{X}_3^{A,R}\right) = e\left(R_{\mathsf{sig}}^1, T_{\mathsf{sig}}^1\right) \ \wedge\ e(S_{\mathsf{sig}}^1, \mathsf{G}_2) = e(\mathsf{G}_1, T_{\mathsf{sig}}^1)$, where $\mathsf{vk}_{\mathsf{sig}}^{A,R} := (\hat{X}_1^{A,R}, \hat{X}_2^{A,R}, \hat{X}_3^{A,R})$ and $\sigma_{\mathsf{sig}}^1 := (R_{\mathsf{sig}}^1, S_{\mathsf{sig}}^1, T_{\mathsf{sig}}^1)$. Thus we use the GS proofs to instantiate this relation in zero-knowledge.

$\blacklozenge$ To prove that the used k in the first relation and the obove relation are the same, the prover use the bridging sigma protocols described in Figure 24. More precisely, it runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \ \wedge\ \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\}$, where cm_1 is obtained via Σ-PRF protocol while cm_2 is computed by the GS proof systems.

$\mathcal{L}_1$**.4.** $\boxed{\mathsf{ct}_{\mathsf{ctr}} = \mathsf{PKE.Enc}(\mathsf{mpk}_{\mathsf{PKE}}^{A}, m_x)}$: To prove the knowledge of a valid ciphertext encrypting the hidden message m_x, we utilize the sigma protocol described in Figure 23 and the prover runs $\Sigma\text{-}\mathsf{ElGamal}\{(m_x, r, e) \mid \mathsf{ct}_1 = \mathsf{G}_1^{r} \ \wedge\ \mathsf{ct}_2 = \mathsf{G}_1^{m_x}(\mathsf{pk}_{\mathsf{PKE}}^{A})^{r} \ \wedge\ \mathsf{cm} = \mathsf{G}_1^{m_x}\mathsf{H}_1^{e}\}$.

$\blacklozenge$ To prove the message m_x in the SPS relation and the obove relation are the same, the prover runs $\Sigma\text{-}\mathsf{Bridging}\{(m_x, e_1, e_2, u_1, 0) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1}\mathsf{K}_1^{u_1} \ \wedge\ \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\}$. In which the commitment cm_1 is obtained via the GS proof of SPS signature σ_{sig}^1, while cm_2 is computed by the sigma protocol Σ-$\mathsf{ElGamal}$.

$\mathcal{L}_1$**.5.** $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}, (\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}}, \mathsf{ID}_{\mathsf{ctr}}), \sigma_{\mathsf{ctr}}) = 1$: The knowledge of a BLS signature on public message $m = (\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}} || \mathsf{ID}_{\mathsf{ctr}})$ signed under the hidden signing key $\mathsf{sk}_{\mathsf{sig}}$ can be written as a PPE of the form, $e\left(\mathsf{vk}_{\mathsf{sig}}, H(\mathsf{vk}_{\mathsf{sig}}^{\mathsf{ctr}} || \mathsf{ID}_{\mathsf{ctr}})\right) = e(\mathsf{G}_1, \sigma_{\mathsf{ctr}})$, where $H(\cdot)$ is a hash-to-curve function as a part of public parameters. We use the GS proofs to instantiate this relation in zero-knowledge.

♦ To show the fact that the verification key, $\mathsf{vk_{sig}}$, used in the above GS proof is already certified by the CA and is identical to the one in the GS of the SPS signature σ^1_{sig}, the prover makes a bridge between the relations discussed in $\mathcal{L}_1.3$ and $\mathcal{L}_1.5$ via proving the following PPE instead with shared commitments [45].

$$e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^{A,R}\right)^1 e\left(\mathsf{vk_{sig}}, \hat{X}_2^{A,R}\right)^1 e\left(\mathsf{G}_1^{m_x}, \hat{X}_3^{A,R}\right)^1 e\left(R^1_{\mathsf{sig}}, T^1_{\mathsf{sig}}\right)^{-1} = 1_{\mathbb{G}_T} \wedge$$
$$e(S^1_{\mathsf{sig}}, \mathsf{G}_2)^1 e(\mathsf{G}_1, T^1_{\mathsf{sig}})^{-1} = 1_{\mathbb{G}_T} \wedge$$
$$e\left(\mathsf{vk_{sig}}, H(\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}} || \mathsf{ID_{ctr}})\right)^1 e(\mathsf{G}_1, \sigma_{\mathsf{ctr}})^{-1} = 1_{\mathbb{G}_T}.$$

This PPE involves both relations in $\mathcal{L}_1.3$ and $\mathcal{L}_1.5$ and we use the same commitment to the group element $\mathsf{vk_{sig}}$ in all relations.

Language $\mathcal{L}_2$. The second language in the ul-PCS with separable policies takes the instance $x_{\mathsf{st}} = \left(\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}^{A,S}_{\mathsf{sig}}, \mathsf{pk}^A_{\mathsf{PKE}}\right)$ and witness $w_{\mathsf{st}} = \left(\mathsf{k}, \mathsf{ctr}, \mathsf{sk}^A_{\mathsf{PKE}}, \sigma^2_{\mathsf{sig}}\right)$ as inputs and then the prover proves the satisfiability of the following relations:

$\mathcal{L}_2$**.1**. $\boxed{\mathsf{ID}_S = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})}$: The prover runs Σ-$\mathsf{PRF}\left\{(\mathsf{k}, \mathsf{ctr}) \mid \mathsf{ID}_S = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\right\}$, depicting the well-formedness of ID_S.

$\mathcal{L}_2$**.2**. $\boxed{\mathsf{DS.Verify}(\mathsf{vk}^{A,R}_{\mathsf{sig}}, (\mathsf{k}, \mathsf{sk}^A_{\mathsf{PKE}}), \sigma^2_{\mathsf{sig}}) = 1}$: The possession of a SPS signature on message $\vec{M} = (\mathsf{G}_1^{\mathsf{k}}, \mathsf{sk}^A_{\mathsf{PKE}})$, signed by the CA can be written as a PPE of the form, $e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^{A,S}\right) e\left(\mathsf{G}_1^{\mathsf{sk}^A_{\mathsf{PKE}}}, \hat{X}_2^{A,S}\right) = e\left(R^2_{\mathsf{sig}}, T^2_{\mathsf{sig}}\right) \wedge e(S^2_{\mathsf{sig}}, \mathsf{G}_2) = e(\mathsf{G}_1, T^2_{\mathsf{sig}})$, where $\mathsf{vk}^{A,S}_{\mathsf{sig}} := (\hat{X}_1^{A,S}, \hat{X}_2^{A,S})$ and $\sigma^2_{\mathsf{sig}} := (R^2_{\mathsf{sig}}, S^2_{\mathsf{sig}}, T^2_{\mathsf{sig}})$. We use the GS proof systems to represent the satisfiability of this PPE.

♦ To demonstrate the fact that the PRF key k used in $\Sigma.\mathsf{PRF}$ and in the above GS proof is indeed the same, the prover runs Σ-$\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$. The commitment cm_1 is obtained via the sigma protocol described in Figure 26 while the commitment cm_2 is computed by the GS proof of knowledge σ^2_{sig} on the satisfiability of the SPS.

$\mathcal{L}_2$**.3**. $\boxed{\mathsf{PKE.Dec}(\mathsf{sk}^A_{\mathsf{PKE}}, \mathsf{ct}_R) = 1}$: The knowledge of a valid secret key $\mathsf{sk}^A_{\mathsf{PKE}}$ such that it can decrypt the receiver's ciphertext ct_R to $m = 1$. To prove this relation in zero-knowledge we use the Dlog sigma protocol described in Figure 22 and the prover runs, Σ-$\mathsf{Dlog}\left\{(\mathsf{sk}^A_{\mathsf{PKE}}) \mid \mathsf{ct}_{R,2}/\mathsf{G}_1 = (\mathsf{ct}_{R,1})^{\mathsf{sk}^A_{\mathsf{PKE}}}\right\}$.

♦ To show the used secret key $\mathsf{sk}^A_{\mathsf{PKE}}$ in the above sigma protocol is the same as the one signed in σ^2_{sig}, the prover runs Σ-$\mathsf{Bridging}\left\{(\mathsf{sk}^A_{\mathsf{PKE}}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{sk}^A_{\mathsf{PKE}}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{sk}^A_{\mathsf{PKE}}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\right\}$, where cm_1 is obtained via Σ-Dlog protocol while the commitment cm_2 is computed by the GS proof systems on the knowledge of SPS signature σ^2_{sig}.

$\mathcal{L}_2$**.4.** $\boxed{\mathsf{sk}^A_{\mathsf{PKE}} \approx \mathsf{vk}^A_{\mathsf{PKE}}}$: The prover to show the knowledge of the secret key $\mathsf{sk}^A_{\mathsf{PKE}}$ and the fact that it corresponds to the public encryption key $\mathsf{pk}^A_{\mathsf{PKE}}$, uses the sigma protocol described in Figure 22 and runs $\Sigma\text{-}\mathsf{Dlog}\left\{(\mathsf{sk}^A_{\mathsf{PKE}}) \mid \mathsf{pk}^A_{\mathsf{PKE}} = \mathsf{G}_1^{\mathsf{sk}^A_{\mathsf{PKE}}}\right\}$.

$\blacklozenge$ The prover runs the bridging sigma protocol $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{sk}^A_{\mathsf{PKE}}, e_1, e_2, 0, 0) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{sk}^A_{\mathsf{PKE}}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{sk}^A_{\mathsf{PKE}}}\mathsf{H}_2^{e_2}\}$ to prove the used secret key $\mathsf{sk}^A_{\mathsf{PKE}}$ in the above relations are the same. The commitment cm_1 is obtained by the sigma protocol $\Sigma\text{-}\mathsf{Dlog}$ of the PKE's decryption correctness while the commitment cm_2 is computed by sigma protocol $\Sigma\text{-}\mathsf{Dlog}$ to show the relation of PKE's public and secret keys.

D.3 ul-PCS for RBAC Policies

Language $\mathcal{L}_1$. The first language in the RBAC ul-PCS takes the instance $x_{\mathsf{st}} = \left(T_{\mathsf{Rand}}, \mathsf{ID}_{\mathsf{ctr}}, \mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \vec{M} := (\mathsf{A}'_1, \mathsf{A}'_2, \mathsf{G}'_2), \mathsf{vk}^A_{\mathsf{sig}}\right)$ and witness $w_{\mathsf{st}} = \left(\mathsf{k}, \mathsf{ctr}, \mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, w_{\mathsf{k}}, \sigma^1_{\mathsf{sig}}, \sigma_{\mathsf{sig}}\right)$ as inputs and the prover proves the satisfiability of the following relations:

$\mathcal{L}_1$**.1.** $\boxed{\mathsf{ACC.MemVrf}(\mathsf{A}'_1, \mathsf{k}, w_{\mathsf{k}}) = 1}$: To prove the possession of a hidden membership witness w_{k} that verifies the accumulator value A'_1 the prover uses the GS proof systems. The satisfiability of the verification of the given accumulator scheme can be written as a PPE of the form, $e\left(w_{\mathsf{k}}, \mathsf{A}'_1\right) e\left(w_{\mathsf{k}}, (\mathsf{G}'_2)^{\mathsf{k}}\right) = e\left(\mathsf{G}_1, \mathsf{G}'_2\right)$. We use the GS proofs to prove the satisfiability of this equation in zero-knowledge.

$\mathcal{L}_1$**.2.** $\boxed{\mathsf{ID}_{\mathsf{ctr}} = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})}$: We use the sigma protocol described in Figure 26 to prove the well-formedness of DY PRF, i.e. $\Sigma\text{-}\mathsf{PRF}\{(\mathsf{k}, \mathsf{ctr}) \mid \mathsf{ID}_{\mathsf{ctr}} = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\}$ over cyclic group $\mathbb{G}_1$.

$\blacklozenge$ The prover to make a bridging between the above relations and showing the fact that the used PRF key k in the both of them is the same secret witnees runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{ctr}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{ctr}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{ctr}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$. In which the commitment cm_1 is obtained via $\Sigma\text{-}\mathsf{PRF}$ protocol while the commtiment cm_2 is computed in the GS proof on the satisfiability of the accumulator verification algorithm.

$\mathcal{L}_1$**.3.** $\mathsf{ctr} < T_{\mathsf{Rand}}$: Additionally, the prover utilizes the range-proof techniques to prove $\mathsf{ctr} \in [0, T_{\mathsf{Rand}})$.

$\blacklozenge$ The prover runs $\Sigma\text{-}\mathsf{Bridging}\{(\mathsf{ctr}, e_1, e_2, 0, 0) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{ctr}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{ctr}}\mathsf{H}_2^{e_2}\}$ to prove the used hidden counter ctr in the above relations is the same. In which the commitment cm_1 is obtained via $\Sigma\text{-}\mathsf{PRF}$ protocol while the commtiment cm_2 is computed in the range-proof protocol.

$\mathcal{L}_1$**.4.** $\boxed{\mathsf{DS.Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, \mathsf{vk}_{\mathsf{sig}}), \sigma^1_{\mathsf{sig}}) = 1}$: To prove the verification phase of the SPS signature σ^1_{sig} satisfies under message $\vec{M} = (\mathsf{G}_1^{\mathsf{k}}, \mathsf{vk}_{\mathsf{sig}})$ and the fact that it is signed by the CA, we can show it via a PPE of the form, $e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}^A_1\right) e\left(\mathsf{vk}_{\mathsf{sig}}, \hat{X}^A_2\right) = e\left(R^1_{\mathsf{sig}}, T^1_{\mathsf{sig}}\right) \wedge e(S^1_{\mathsf{sig}}, \mathsf{G}_2) = e(\mathsf{G}_1, T^1_{\mathsf{sig}})$, where $\mathsf{vk}^A_{\mathsf{sig}} := (\hat{X}^A_1, \hat{X}^A_2)$ and $\sigma^1_{\mathsf{sig}} := (R^1_{\mathsf{sig}}, S^1_{\mathsf{sig}}, T^1_{\mathsf{sig}})$. We use GS proof systems to show the satisfiability of this equation.

♦ To demonstrate that the same k in the first relation and the above relation is used, the prover makes a bridge between them by running Σ-$\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$, where the commitment cm_1 is obtained via Σ-PRF protocol while the commitment cm_2 is computed by the GS proof system on the validity of σ^1_{sig}.

$\mathcal{L}_1$**.5**. $\mathsf{DS.Verify}(\mathsf{vk}_{\mathsf{sig}}, (\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}}, \mathsf{ID}_{\mathsf{ctr}}), \sigma_{\mathsf{ctr}}) = 1$: To validate a newly generated verification key and to bind it with the new identifier $\mathsf{ID}_{\mathsf{ctr}}$, the prover needs to prove the satisfiability of a PPE relation described as, $e\left(\mathsf{vk}_{\mathsf{sig}}, H(\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}} || \mathsf{ID}_{\mathsf{ctr}})\right) = e(\mathsf{G}_1, \sigma_{\mathsf{ctr}})$ that represents the validity of BLS signature. We use GS proof systems to instantiate this relation in zero-knowledge.

♦ To show the fact that the verification key, $\mathsf{vk}_{\mathsf{sig}}$, used in the above GS proof is already certified by the CA and is identical to the one in the GS of the SPS signature σ^1_{sig}, the prover makes a bridge between the relations discussed in $\mathcal{L}_1.4$ and $\mathcal{L}_1.5$ via proving the following PPE instead.

$$
\begin{aligned}
&e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^A\right)^1 e\left(\mathsf{vk}_{\mathsf{sig}}, \hat{X}_2^A\right)^1 e\left(R^1_{\mathsf{sig}}, T^1_{\mathsf{sig}}\right)^{-1} = 1_{\mathbb{G}_T} \wedge \\
&e(S^1_{\mathsf{sig}}, \mathsf{G}_2)^1 e(\mathsf{G}_1, T^1_{\mathsf{sig}})^{-1} = 1_{\mathbb{G}_T} \wedge \\
&e\left(\mathsf{vk}_{\mathsf{sig}}, H(\mathsf{vk}^{\mathsf{ctr}}_{\mathsf{sig}} || \mathsf{ID}_{\mathsf{ctr}})\right)^1 e(\mathsf{G}_1, \sigma_{\mathsf{ctr}})^{-1} = 1_{\mathbb{G}_T} .
\end{aligned}
$$

This PPE involves both relations in $\mathcal{L}_1.4$ and $\mathcal{L}_1.5$ with a single commitment to $\mathsf{vk}_{\mathsf{sig}}$.

Language $\mathcal{L}_2$. In the second relation, the prover takes the instance $x_{\mathsf{st}} = (\mathsf{ID}_S, \mathsf{ct}_R, \mathsf{vk}^A_{\mathsf{sig}}, \mathsf{pp}', \mathsf{A}')$ and the witness $w_{\mathsf{st}} = (\mathsf{k}, \mathsf{ctr}, x, w, \sigma^2_{\mathsf{sig}})$ as input and acts as follows:

$\mathcal{L}_2$**.1**. $\boxed{\mathsf{ID}_S = \mathsf{PRF.Eval}(\mathsf{k}, \mathsf{ctr})}$: To prove the well-formedness of the PRF evaluation the prover runs the sigma protocol Σ-$\mathsf{PRF}\{(\mathsf{k}, \mathsf{ctr}) \mid \mathsf{ID}_S = \mathsf{G}_1^{1/(\mathsf{k}+\mathsf{ctr})}\}$ over the cyclic group $\mathbb{G}_1$.

$\mathcal{L}_2$**.2**. $\mathsf{DS.Verify}(\mathsf{vk}^A_{\mathsf{sig}}, (\mathsf{k}, w), \sigma^2_{\mathsf{sig}}) = 1$: This relation can be formulated by a PPE of the form, $e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^A\right) e\left(w, \hat{X}_2^A\right) = e\left(R^2_{\mathsf{sig}}, T^2_{\mathsf{sig}}\right) \wedge e(S^2_{\mathsf{sig}}, \mathsf{G}_2) = e(\mathsf{G}_1, T^2_{\mathsf{sig}})$, where $\mathsf{vk}^A_{\mathsf{sig}} := (\hat{X}_1^A, \hat{X}_2^A)$ and $\sigma^2_{\mathsf{sig}} := (R^2_{\mathsf{sig}}, S^2_{\mathsf{sig}}, T^2_{\mathsf{sig}})$ and the prover can prove the satisfiability of the relation by GS proof systems.

♦ The prover runs Σ-$\mathsf{Bridging}\{(\mathsf{k}, e_1, e_2, 0, u_2) \mid \mathsf{cm}_1 = \mathsf{G}_1^{\mathsf{k}}\mathsf{H}_1^{e_1} \wedge \mathsf{cm}_2 = \mathsf{G}_2^{\mathsf{k}}\mathsf{H}_2^{e_2}\mathsf{K}_2^{u_2}\}$ to prove the fact that the PRF key k used in $\Sigma.\mathsf{PRF}$ and is already signed by the CA. The commitment cm_1 is obtained via the sigma protocol described in Figure 26 while the commitment cm_2 is computed by the GS proof of knowledge σ^2_{sig}.

$\mathcal{L}_2$**.3**. $\mathsf{ACC.MemVrf}(\mathsf{A}', x, w) = 1$: Similar to the previous languages, the prover can describe the membership verification of the accumulator scheme by the satisfiability of a PPE of the form, $e(w, \mathsf{A}')\, e\left(w, (\mathsf{G}_2')^x\right) = e(\mathsf{G}_1, \mathsf{G}_2')$. Thus it runs the GS proof to show the possession of hidden parameters.

♦ The prover bridges the relations describe in $\mathcal{L}_2.2$ and $\mathcal{L}_2.3$ to show the fact that the membership witness w which passes the accumulator verification is already certifies and is signed in SPS signature σ^2_{sig}. For this aim the prover proves the following PPE instead:

$$
\begin{aligned}
& e\left(\mathsf{G}_1^{\mathsf{k}}, \hat{X}_1^A\right)^1 e\left(w, \hat{X}_2^A\right)^1 e\left(R^2_{\mathsf{sig}}, T^2_{\mathsf{sig}}\right)^{-1} = 1_{\mathbb{G}_T} \wedge \\
& e(S^2_{\mathsf{sig}}, \mathsf{G}_2)^1 e(\mathsf{G}_1, T^2_{\mathsf{sig}})^{-1} = 1_{\mathbb{G}_T} \wedge \\
& e\left(w, \mathsf{A}'\right)^1 e\left(w, (\mathsf{G}_2')^x\right)^1 e\left(\mathsf{G}_1, \mathsf{G}_2'\right)^{-1} = 1_{\mathbb{G}_T} \cdot
\end{aligned}
$$

E Distributed Setup and KeyGen Algorithms

In the following, we showcase that using standard techniques we can achieve distributed implementations of the algorithms Setup and KeyGen for our three constructions. We assume an honest-but-curious model for the sake of the argument, however a lifting to malicious security would again follow standard techniques.

E.1 The Generic ul-PCS Scheme

First, we look at how the generic ul-PCS, proposed in Figure 14, and its concrete instantiation based on the OT12's inner product predicate encryption [55] can be distributed. Recall that the CA holds the predicate encryption's master secret key, $\mathsf{msk}_{\mathsf{PE}}$, along with a signature key $\mathsf{sk}^A_{\mathsf{sig}}$. On the other hand, a user keeps will obtain the PRF seed k and a predicate encryption secret key sk_f along with its root signature key-pair $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{vk}_{\mathsf{sig}})$. To generate these secret elements in a distributed manner, we can follow the following steps:

1. *CA-side setup:*
   (a) Given the description of the OT12 IP-PE scheme in Appendix C, we can generate the keys in a distributed way, where each server holds a share B_i^* of the matrix $B^* = \prod_i B_i^*$ (component-wise product). This could be done with a standard MPC, and essentially, we need a sum-sharing of a matrix X and its inverse Y. In particular, this means each entry Y_{ij} is shared among n distinct certificate authorities. While this is a heavier computation, implementations of such an operation based on the methods by Blom et al. are possible [12].
   (b) Each CA possesses its own signature key-pair.
2. *Registration of a client for attributes x:*
   (a) Each CA samples random seed k_i and a public key share vk_i.
   (b) The CAs create additional shared randomness in anticipation of the creation of the secret functional key. They compute a sum-sharing of $n+1$ random elements r_k.
   (c) Then they run a multiplication protocol to obtain a sharing of selected matrix elements: $r_1 Y_{i,2}, \ldots, r_1 Y_{i,n+1}$ (those are the positions for generating a scaled vector of attributes) and $r_j Y_{i,3n+j} (1 < j \leq n+1)$ (those are the positions where random exponents are needed).
   (d) Recalling from Appendix C.1 that in OT12 the functional key sk_f is a vector whose ith component is $\prod_{j=1}^N \mathsf{G}_1^{Y_{ij} z_j}$, we observe that each CA can compute a meaningful share sk_f^i by doing this computation based on the attribute x and the sharing of elements Y_{ij} resp. $r_k Y_{ij}$ (for those indices where additional randomness is needed).

(e) The CA signs the pairs $(\mathsf{k}_i, x), (\mathsf{k}, \mathsf{sk}_f^i)$, and $(\mathsf{k}, \mathsf{vk}_i)$.
(f) *Aggregation step:*
   i. Functional key shares are aggregated by component-wise multiplication of the vectors sk_f^i — The addition in the exponent leads to the expression in (d) as everything has been computed as a sum-sharing.
   ii. Seed shares are summed up $\mathsf{k} = \sum_i \mathsf{k}_i$.
   iii. Root key pairs are summed up as well (e.g. assuming a simple DL-based signature scheme).

3. Finally, the aggregated pairs $(\mathsf{k}, \mathsf{sk}_f), (\mathsf{k}, x)$ and $(\mathsf{k}, \mathsf{vk}_{\mathsf{sig}})$ are certifiable, because in any of the languages $\mathcal{L}_1$ and $\mathcal{L}_2$, instead of proving the knowledge of a signature from the CA, one would have to prove the aggregation is done correctly based on the signed shares by each CA. While conceptually possible, by an additional round of interaction, one can even shift more computational overhead to the registration phase as outlined below:

3'. Alternatively to the above certification, one can add one round of interaction, where the client commits to each aggregated pair, proves the well-formedness using a NIZK and obtains a threshold signature on the commitments. In this case, each pair is certifiable in the first and second NIZK languages by having one additional commitment plus a signature on it. In this case, for further efficiency, the utilized SPS scheme can be replaced with the recent Threshold SPS-scheme of Crites et al. [27]. In this case, each CA has its own SPS signature key-pair and a sufficiently large number of issuers is needed to obtain a valid signature (with respect to the aggregated public key). This strategy pushes most of the computational overhead into the registration phase.

Furthermore, simplifications can be made depending on the adversarial model. We observe that the root key pair $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{vk}_{\mathsf{sig}})$ for the party is never revealed by the party in any operation, and thus we can simply let one of the servers decide for that one if we are in an honest-but-curious setting.

E.2 The ul-PCS with Separable Policies

Similarly, we can distribute the generation of the secret keys in the ul-PCS scheme with separable policies. In this scheme, the PE scheme is "realized" using ordinary PKE with keys $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}})$. Furthermore, we have signature keys to authorize sender and receiver predicates. Compared to the generic scheme discussed above, it is much simpler and we can run a distributed-key generation in advance and each CA has its own signature key-pair. We briefly discuss how the user's registration works concretely.

1. *CA-side setup:* Each CA samples random PRF seed $\mathsf{k}_i \xleftarrow{\$} \mathbb{Z}_p^*$ and a public key share vk_i.
2. *Registration of a client for attributes x:* Each CA issues a signature on the value $(\mathsf{k}_i, \mathsf{vk}_i, m)$, where $m \in \{0,1\}$ is a bit. If $m = 1$, the client also gets a signature on $(\mathsf{k}_i, \mathsf{sk}_i)$.
   The client performs standard aggregation to compute all relevant values ($\mathsf{sk}_{\mathsf{sig}}$ from all sk_i's, k from all k_i's).
3. Certification is again possible via a NIZK, or via one more round of interaction as above.

E.3 The Role-based ul-PCS Construction

The signing process in the role-based ul-PCS is as above, but the relevant values that could break privacy are the accumulator witnesses (because they would allow to test which attributes can send to a target public key), and the seed values. Hence, here one has to do the following:

1. CA setup: As the accumulator witnesses in this case are just signatures on roles that belong to an accumulator value, we just set up a threshold signature scheme. Each CA then holds a signature share on a role i for accumulator A (identified by the signature public key).
2. Registration of a client for attributes x:
   (a) Each CA samples random PRF seed k_i and a public key share vk_i.
   (b) The user can simply obtain the partial signature shares and a combination of them and finally is in possession of the full witness for its role x.
   (c) The remaining steps are as above: the client can reconstruct the full seed k_i, the full root signature keypair $(\mathsf{vk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}})$, and has all witnesses.
3. Certification can be done via a NIZK or via another round of interaction as above.

F Preliminaries on One-Time Accounts (OTA)

An OTA scheme [31] is defined as a tuple of algorithms $\mathsf{OTA} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{NoteGen}, \mathsf{Enc}, \mathsf{Receive}, \mathsf{NulEval})$ with the following syntax and intended semantics:

- Setup: Generates the public parameters that is given implicitly to any algorithm below as input.
- KeyGen: Generates an asymmetric key-pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{NoteGen}(\mathsf{pk}, \vec{a}; r)$: Takes a public key and a vector of type-value pairs and generates the note, i.e. the account.
- $\mathsf{Enc}(\mathsf{pk}, (\vec{a}, r))$: Encrypts the information toward the recipient such that the recipient will be able to reconstruct the note's content and to spend it (see below) .
- $\mathsf{Receive}(\mathsf{note}, C, \mathsf{sk})$: If the note and ciphertext are created for the public key belonging to sk, then the algorithm returns the values $(\vec{a}, r)$, and otherwise returns $\perp$.
- $\mathsf{NulEval}(\mathsf{sk}, r)$: Returns the nullifier value that is tied to a particular note (generated with randomness r). The nullifier is needed to spend the tokens contained in a note.

OTA's must be accompanied by some efficient NIZK languages, including the ones we need in our construction in Section 7, which are shown to be efficiently realizable [31] using for example Groth-Sahai proof systems. The security requirements from an OTA scheme include: (1) Nullifiers should appear pseudo-random and be unique, such that they can be presented as evidence of spending a coin, and double spends would directly visible by repeated nullifiers, (2) the note is binding to the key and values, unique, as well as private in that it hides its content. We discuss these requirements in the security analysis of our extended scheme.