An extended abstract of this paper appears in the proceedings of PKC 2014. This is the full version, available as IACR Cryptology ePrint Archive Report 2013/413.

Policy-Based Signatures

Mihir Bellare* Georg Fuchsbauer†

Abstract

We introduce policy-based signatures (PBS), where a signer can only sign messages conforming to some authority-specified policy. The main requirements are unforgeability and privacy, the latter meaning that signatures not reveal the policy. PBS offers value along two fronts: (1) On the practical side, they allow a corporation to control what messages its employees can sign under the corporate key. (2) On the theoretical side, they unify existing work, capturing other forms of signatures as special cases or allowing them to be easily built. Our work focuses on definitions of PBS, proofs that this challenging primitive is realizable for arbitrary policies, efficient constructions for specific policies, and a few representative applications.

Keywords: Signatures, policies, group signatures, NIZKs, Groth-Sahai proofs, delegation.

1 Introduction

PBS. In a standard digital signature scheme [DH76, GMR88], a signer who has established a public verification key vk and a matching secret signing key sk can sign any message that it wants. We introduce policy-based signatures (PBS), where a signer's secret key sk_p is associated to a policy $p \in \{0,1\}^*$ that allows the signer to produce a valid signature σ of a message m only if the message satisfies the policy, meaning (p, m) belongs to a *policy language* $L \subseteq \{0,1\}^* \times \{0,1\}^*$ associated to the scheme.

This cannot be achieved if the signer creates her keys in a standalone way. In our model, a signer is issued a signing key sk_p for a particular policy p by an authority, as a function of a master secret key msk held by the authority. Verification that σ is a valid signature of m is then done with respect to the authority's public parameters pp.

Within this framework, we consider a number of security goals. The most basic are unforgeability and privacy. Unforgeability says that producing a valid signature for message m is infeasible unless one has a secret key sk_p for some policy p such that $(p, m) \in L$. (You can only sign messages that you are allowed to sign.) Privacy says that signatures do not reveal the policy under which they were created. We will propose and explore different formalizations of these goals.

A trivial way to achieving PBS is via certificates. In more detail, to issue a secret key sk_p for policy p, the authority generates a fresh key pair (sk, pk) for an ordinary signature scheme, creates a certificate $cert$ consisting of a signature of (p, pk) under the authority's signing key msk, and returns $sk_p = (sk, pk, p, cert)$ to the signer. The latter's signature on m is now an ordinary signature of m under sk together with $(pk, p, cert)$, and verification is possible given the public verifying key pp of the authority. However, while this will provide unforgeability, it does not provide privacy, because the policy must be revealed in the signature to allow for verification. Similarly, privacy in the absence of unforgeability is also trivial. The combination of the two requirements, however, results in a non-trivial goal.

* Department of Computer Science and Engineering, University of California San Diego, La Jolla CA 92093. Supported in part by NSF grants CNS-1228890, CNS-1116800, CNS-0904380 and CCF-0915675.

† Institute of Science and Technology Austria. Supported by the European Research Council, ERC Starting Grant (259668-PSPC); part of his work was done while at Bristol University, supported by EPSRC grant EP/H043454/1.

PBS may be viewed as an authentication analogue of functional encryption [BSW11]. We can view the latter as allowing decryption to be policy-restricted rather than total, an authority issuing decryption keys in a way that enforces the policy. Correspondingly, in PBS the signing capability is policy-restricted rather than total, an authority issuing signing keys in a way that enforces the policy.

Why PBS? Given that there already exist many forms of signatures, one might ask why another. PBS offers value along two fronts, practical and theoretical. On the practical side, the setup of PBS is natural in a corporate or other hierarchical environment. For example, a corporation may want to allow employees to sign under the company public key pp, but may want to restrict the signing capability of different employees based on their positions and privileges. However, the company policies underlying the restrictions need to be kept private. On the theoretical side, PBS decreases rather than increases complexity in the area because it serves as an umbrella notion that unifies existing notions by capturing some as special cases and allows others to be derived in simple and natural ways. In particular, this is true for a significant body of work on signatures that have privacy features, including group signatures [Cv91, BMW03], proxy signatures [MUO96], ring signatures [RST01, BKM06], mesh signatures [Boy07], anonymous proxy signatures [FP08], attribute-based signatures [MPR11] and anonymous credentials [CL01, BCKL08].

Policy languages. We wish to allow policies as expressive and general as possible. We accordingly allow the policy language to be any language in **P**, which captures most typical applications, where one can test in polynomial time whether a given policy allows a given message. At first this may seem as general as one can get, but we go further, allowing the policy language to be any language in **NP**. This means that the policies that can be expressed and enforced are restricted neither in form nor type, the only condition being that, given a witness, one can test in polynomial time whether a policy allows a given message. We will see applications where it is important that policy languages can be in **NP** rather than merely in **P**.

Definitions and relations. We first provide an unforgeability definition and an indistinguishability-based privacy definition. Unforgeability says that an adversary cannot create a valid signature of a message m without having a key for some policy p such that $(p, m) \in L$, even when it can obtain keys for other policies, and signatures for other messages under the target policy. Indistinguishability says that the verifier cannot tell under which of two keys a signature was created assuming both policies associated to the keys permit the corresponding message. Our definition also implies that the verifier cannot decide whether two signatures were created using the same key.

However, indistinguishability may not always provide privacy. For example, if for each message m there is only one policy p_m such that $(p_m, m) \in L$ then even a scheme where a signature of m reveals p_m satisfies indistinguishability. We provide a stronger privacy notion, called simulatability, that says that real signatures look like ones a simulator could generate without knowledge of the policy or any key derived from the policy. This strong notion of privacy is not subject to the above-discussed weaknesses of indistinguishability. The situation parallels that for functional encryption (FE), where an indistinguishability-based requirement was shown to not always suffice [BSW11, O'N10] and stronger simulatability requirements have been defined and considered [BSW11, O'N10, BO13, DCIJ$^+$13, AGVW13, BF13, MM13]. However, for FE, impossibility results show that the strongest and most desirable simulation-based definitions are not achievable [BSW11, BO13, DCIJ$^+$13, AGVW13, MM13]. In contrast, for PBS we will show that our simulatability notion is achievable in the standard model under standard assumptions.

We also strengthen unforgeability to provide an extractability notion for PBS. We show that simulatability implies indistinguishability, and simulatability+extractability implies unforgeability. Simulatability+extractability emerges as a powerful security notion that enables a wide range of applications.

Constructions. PBS for arbitrary **NP** policy languages achieving simulatability+extractability is an ambitious target. The first question that emerges is whether this can be achieved, even in principle, let alone efficiently. We answer in the affirmative via two generic constructions based on standard primitives. The first uses ordinary signatures, IND-CPA encryption and standard non-interactive zero-knowledge (NIZK) proofs. The second uses only ordinary signatures and simulation(-sound) extractable NIZK proofs [Gro06].

While our generic constructions prove the theoretical feasibility of PBS, their use of general NIZKs makes them inefficient. We ask whether more efficient solutions may be given without resorting to the random-oracle model [BR93]. We combine Groth-Sahai proofs [GS08] and structure-preserving signatures [AFG+10] to design efficient PBS schemes for policy languages expressible via equations over a bilinear group. This construction requires a twist over usual applications of Groth-Sahai proofs, namely, in order to hide the policy, we swap the roles of constants and variables. This provides a tool that, like structure-preserving signatures, is useful in cryptographic applications where policies may be about group elements.

Applications and implications. We illustrate applicability by showing how to derive a variety of other primitives from PBS in simple and natural ways. This shows how PBS can function as a unifying framework for signatures and beyond. In Section 5.1 we show that PBS implies group signatures meeting the strong CCA version of the definition of [BMW03]. We also show that PBS implies attribute-based signatures, as modeled by [MPR11] (Appendix D). In Section 5.2 we show that PBS also implies signatures of knowledge [CL06]. These applications are illustrative rather than exhaustive, many more being possible.

Our generic constructions discussed above show which primitives are sufficient to build PBS. A natural question is which primitives are necessary, namely, which fundamental primitives are implied by PBS? In Section 5.2, we address this and show that PBS implies seemingly unrelated primitives like IND-CPA encryption and simulation-extractable NIZK proofs [Gro06]. By [Sah99] this means PBS implies IND-CCA encryption. In particular, this means the assumptions we make for our generic constructions are not only sufficient but necessary.

Delegatable PBS. In Section 6 we extend the PBS framework to allow delegation. This means that an entity receiving from the authority a key sk_{p_1} for a policy p_1 can then issue to another entity a key $sk_{p_1 \| p_2}$ that allows the signing of messages m which satisfy both policies p_1 and p_2. This can continue, with the holder of $sk_{p_1 \| p_2}$ further delegating a key $sk_{p_1 \| p_2 \| p_3}$, and so on. This is useful in a hierarchical setting, where a company president can delegate to vice presidents, who can then delegate to managers, and so on. We provide definitions which extend and strengthen those for the basic PBS setting; in particular, privacy must hold even when the adversary chooses the user keys. We then show how to achieve delegatable PBS for policy chains of arbitrary polynomial length. As with basic PBS, certificate chains yield a trivial construction if privacy is not required, but when it is the problem is non-trivial. For simplicity, we base our construction, achieving sim+ext security, on append-only signatures [KMPR05], which can however be easily constructed from ordinary signatures.

Discussion. In the world of digital signatures, extensions of functionality typically involve some form of delegation of signing rights: group signatures allow members to sign on behalf of a whole group, in attribute-based signatures (ABS) and types of anonymous credentials, keys are also issued by an authority, and (anonymous) proxy signatures model delegation and re-delegation explicitly. For most of these primitives, anonymity or privacy notions have been considered. A group signature, for example, should not reveal which group member produced a signature on behalf of the group (while an authority can trace group signatures to their signer). In ABS, users hold keys corresponding to their attributes and can sign messages with respect to a policy, which is a predicate over attributes. Users should only be able make signatures for policies satisfied by their attributes. Privacy for ABS means that a signature should reveal nothing about the attributes of the key under which it was produced, other than the fact that it satisfies the policy.

In the models of primitives such as attribute-based signatures or mesh signatures, the policy itself is always public, as is the warrant that specifies the policy in (even anonymous) proxy signatures. With PBS, we ask whether this is a natural limitation of privacy notions, and whether it is inherently unavoidable that objects like the policy (which specify *why* the message could be signed) need to be public.

Consider the example of a company implementing a scheme where each employee gets a signing key and there is one public key which is used by outsiders to verify signatures in the name of the company. A group-signature scheme would allow every employee holding a key to sign on behalf of the company, but there is no fine-grained control over who is allowed to sign which documents. This can be achieved using attribute-based signatures, where each user is assigned attributes, and a message is signed with respect

to a policy like (*CEO* `or` (*board member* `and` *general manager*)). However, it is questionable whether a verifier needs to know the company-internal policy used to sign a specific message, and there is no apparent reason he should know; all he needs to be assured of is that the message was signed by someone entitled to, but not who this person is, what she is entitled to sign, nor whether two messages were signed by the same person. This is what PBS provides.

Another issue is that when using ABS we have to assume that the verifier can tell which messages can be signed under which policies. An attribute-based signature which is valid under the policy (*CEO* `or` *intern*) tells a verifier that it could have been produced by an intern, but it does not provide any guarantees as to whether an intern would have been entitled to sign the message. We ask whether it is possible to avoid having these types of public policies at all. PBS answers this in the affirmative.

Related work. The use of NIZKs for signatures begins with [BG89], who built an ordinary signature scheme from a NIZK, a pseudorandom function (PRF) and a commitment scheme. Encryption and ordinary signatures were combined with NIZKs to create group signatures in [BMW03]. Our first generic construction builds on these ideas. Our second generic construction, inspired by [DHLW10, BMT13], exploits the power of simulation-extractable NIZKs to give a conceptually simpler scheme that, in addition to the NIZK, uses only an ordinary signature scheme.

In independent and concurrent work, Boyle, Goldwasser and Ivan (BGI) [BGI13] introduce functional signatures, where an authority can provide a key for a function f that allows the signing of any message in the range of f. This can be captured as a special case of PBS in which the policy is f and the policy language is the set of all (f, m) such that m is in the range of f, a witness for membership being a pre-image of m under f. BGI define unforgeability and an indistinguishability-based privacy requirement, but not the stronger simulatability or extractability conditions that we define and achieve. BGI have a succinctness condition which we do not have.

A related primitive is malleable signatures, introduced by Chase, Kohlweiss, Lysyanskaya and Meiklejohn [CKLM13]. They are defined with respect to a set of functions $\mathcal{F}$, so that given a signature of m, anyone can derive a signature of $f(m)$ for $f \in \mathcal{F}$. Concurrently to our work, Backes, Meiser and Schröder [BMS13] introduced delegatable functional signatures, but in their model delegatees have public keys and signatures are verified under the authority's and the delegatee's keys. Privacy means that signatures from delegatees are indistinguishable from signatures from the authority.

Three recent works independently and concurrently introduce PRFs where one may issue a key to evaluate the PRF on a subset of the points of the domain [BW13, BGI13, KPTZ13]. These can be viewed as PRF analogues of policy-based signatures in which a policy corresponds to a set of inputs and a key allows computation of the PRF on the inputs in the set. Boneh and Waters [BW13] also provide a policy-based key-distribution scheme.

In their treatment of policy-based cryptography, Bagga and Molva [BM05] mention both policy-based encryption and policy-based signatures. However they do not consider privacy, without which, as noted above, the problem is easy. Moreover, they have no formal definitions of security requirements or proofs that their bilinear-map-based schemes achieve any well-defined security goal.

2 Preliminaries

Notations and conventions. The empty string is denoted by ε. If S is a finite set then $|S|$ denotes its size and $s \leftarrow\!\!\$\, S$ denotes picking an element uniformly from S and assigning it to s. For $i \in \mathbb{N}$ we let $[i] = \{1, \ldots, i\}$. We denote by $\lambda \in \mathbb{N}$ the security parameter and by 1^λ its unary representation. Algorithms are randomized unless otherwise indicated. "PT" stands for "polynomial-time," for both randomized and deterministic algorithms. By $y \leftarrow A(x_1, \ldots; R)$, we denote the operation of running algorithm A on inputs $x_1, \ldots$ and coins R and letting y denote the output. By $y \leftarrow\!\!\$\, A(x_1, \ldots)$, we denote letting $y \leftarrow A(x_1, \ldots; R)$ with R chosen at random. We denote by $[A(x_1, \ldots)]$ the set of points that have positive probability of being output by A on inputs $x_1, \ldots$. Adversaries are algorithms or tuples of algorithms. In the latter case, the running time of the adversary is the sum of the running times of the algorithms in the tuple.

A map $\mathsf{R}\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is said to be an **NP**-relation if it is computable in time polynomial in the length of its first input. For $x \in \{0,1\}^*$ we let $\mathsf{WS}_{\mathsf{R}}(x) = \{w : \mathsf{R}(x,w) = 1\}$ be the *witness set* of x. We let $\mathcal{L}(\mathsf{R}) = \{x : \mathsf{WS}_{\mathsf{R}}(x) \neq \emptyset\}$ be the *language* associated to R. The fact that R is an **NP**-relation means that $\mathcal{L}(\mathsf{R}) \in \mathbf{NP}$.

Game-playing framework. For our security definitions and proofs we use the code-based game-playing framework of [BR06]. A game **Exp** (Figure 1, for example) consists of a finite number of procedures. We execute a game with an adversary $\mathcal{A}$ and security parameter $\lambda \in \mathbb{N}$ as follows. The adversary gets 1^λ as input. It can then query game procedures. Its first query must be to Initialize with argument 1^λ, and its last to Finalize, and these must be the only queries to these oracles. In between it can query the other oracles as it wishes. The output of the execution, denoted $\mathbf{Exp}_{\mathcal{A}}(\lambda)$ is the output of Finalize. We denote by $\mathbf{Exp}_{\mathcal{A}}(\lambda) \Rightarrow y$ the event that this output is y. In code, boolean flags are assumed initialized to false, sets to $\emptyset$, integers to 0 and array entries to $\perp$. The running time of the adversary $\mathcal{A}$ is a function of λ in which oracle calls are assumed to take unit time.

3 Policy-Based Signatures

Policy languages. A *policy checker* is an **NP**-relation $\mathsf{PC}\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. The first input is a pair (p,m) representing a policy $p \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$, while the second input is a witness $w \in \{0,1\}^*$. The associated language $\mathcal{L}(\mathsf{PC}) = \{(p,m) : \mathsf{WS}_{\mathsf{PC}}((p,m)) \neq \emptyset\}$ is called the *policy language* associated to PC. That $(p,m) \in \mathcal{L}(\mathsf{PC})$ means that signing m is permitted under policy p. We say that (p,m,w) is PC-valid if $\mathsf{PC}((p,m),w) = 1$.

PBS schemes. A *policy-based signature scheme* $\mathcal{PBS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is a 4-tupe of PT algorithms:

(1) Setup: On input the unary-encoded security parameter 1^λ, setup algorithm Setup returns public parameters pp and a master secret key msk.

(2) KeyGen: On input msk and p, where $p \in \{0,1\}^*$ is a policy, key-generation algorithm KeyGen outputs a signing key sk for p.

(3) Sign: On input sk, m and w, where $m \in \{0,1\}^*$ is a message and $w \in \{0,1\}^*$ is a witness, signing algorithm Sign outputs a signature σ.

(4) Verify: On input pp, m and σ, verification algorithm Verify outputs a bit.

We say that the scheme is *correct* relative to policy checker PC if for all $\lambda \in \mathbb{N}$, all PC-valid (p,m,w), all $(pp, msk) \in [\mathsf{Setup}(1^\lambda)]$ and all $\sigma \in [\mathsf{Sign}(\mathsf{KeyGen}(msk,p),m,w)]$ we have $\mathsf{Verify}(pp,m,\sigma) = 1$.

Unforgeability. Our basic unforgeability requirement is that it be hard to create a valid signature of m without holding a key for some policy p such that $(p,m) \in \mathcal{L}(\mathsf{PC})$. The formalization is based on game $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{PBS}}$ in Figure 1. For $\lambda \in \mathbb{N}$ we let $\mathbf{Adv}^{\mathrm{UF}}_{\mathcal{PBS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{PBS},\mathcal{A}} \Rightarrow \mathsf{true}]$. We say that $\mathcal{PBS}$ is *unforgeable*, or UF-*secure*, if $\mathbf{Adv}^{\mathrm{UF}}_{\mathcal{PBS},\mathcal{A}}(\cdot)$ is negligible for every PT $\mathcal{A}$. Via a MakeSK query, the adversary can have the game create a key for a policy p. Then, via Sign, it can obtain a signature under this key for any message of its choice. (This models a chosen-message attack.) It may also, via its RevealSK oracle, obtain the key itself. (This models corruption of users or the formation of collusions of users who pool their keys.) These queries naturally give the adversary the capability of creating signatures for certain messages, namely messages m such that for some p with $(p,m) \in \mathcal{L}(\mathsf{PC})$, it either obtained a key for p or obtained a signature for m. Unforgeability asks that it cannot sign any other messages. Note that we did not explicitly specify how Sign behaves when run on a key for p, and m, w with $\mathsf{PC}((p,m),w) = 0$. However, if it outputs a valid signature, this can be used to break UF-security.

Indistinguishability. Privacy for policy-based signatures requires that a signature not reveal the policy associated to the key and neither the witness that was used to create the signature. A first idea would be the following formalization: an adversary outputs a message m, two policies p_0, p_1, and two witnesses w_0, w_1, such that both (p_0, m, w_0) and (p_1, m, w_1) are PC-valid. For either p_0 or p_1 the experiment computes

$\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{PBS}}$

proc INITIALIZE
$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$; $j \leftarrow 0$
Return pp

proc MAKESK(p)
$j \leftarrow j + 1$; $Q[j][1] \leftarrow p$
$Q[j][2] \leftarrow\!\!\$\ \mathsf{KeyGen}(pp, msk, p)$; $Q[j][3] \leftarrow \emptyset$

proc REVEALSK(i)
If $i \notin [j]$ then return $\bot$
$sk \leftarrow Q[i][2]$; $Q[i][2] \leftarrow \bot$; Return sk

proc SIGN(i, m, w)
If $i \notin [j]$ or $Q[i][2] = \bot$ then return $\bot$
$Q[i][3] \leftarrow Q[i][3] \cup \{m\}$
Return $\mathsf{Sign}(pp, Q[i][2], m, w)$

proc FINALIZE(m, σ)
If $\mathsf{Verify}(pp, m, \sigma) = 0$ then return false
For $i = 1, \ldots, j$ do
  If $(Q[i][1], m) \in \mathcal{L}(\mathsf{PC})$ then
    If $Q[i][2] = \bot$ or $m \in Q[i][3]$ then return false
Return true

$\mathbf{Exp}^{\mathrm{IND}}_{\mathcal{PBS}}$

proc INITIALIZE
$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$; $b \leftarrow\!\!\$\ \{0, 1\}$
Return (pp, msk)

proc LR(p_0, p_1, m, w_0, w_1)
If $\mathsf{PC}((p_0, m), w_0) = 0$ or $\mathsf{PC}((p_1, m), w_1) = 0$
  then return $\bot$
$sk_0 \leftarrow \mathsf{KeyGen}(msk, p_0)$
$sk_1 \leftarrow \mathsf{KeyGen}(msk, p_1)$
$\sigma_b \leftarrow \mathsf{Sign}(sk_b, m, w_b)$
Return (σ_b, sk_0, sk_1)

proc FINALIZE(b')
Return $(b = b')$

Figure 1: Games defining unforgeability and indistinguishability for PBS.

a secret key and uses it to produce a signature on m and gives it to the adversary, who wins if it can guess which policy was used. It turns out that this notion is too weak, as it does not guarantee that two signatures produced under the same secret key do not link, as seen as follows. Consider a scheme satisfying the security notion just sketched and modify it by attaching to each secret key a random string during key generation and alter Sign to append to the signature the random string contained in the secret key. Clearly, two signatures under the same key are linkable, but yet the scheme satisfies the definition. We therefore give the adversary both secret keys in addition to the signature.

Let $\mathbf{Exp}^{\mathrm{IND}}_{\mathcal{PBS},\mathcal{A}}$ be the game defined in Figure 1. We say that $\mathcal{PBS}$ has *indistinguishability* if for all PT adversaries $\mathcal{A}$ we have that $\mathbf{Adv}^{\mathrm{IND}}_{\mathcal{PBS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{IND}}_{\mathcal{PBS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}] - \frac{1}{2}$ is negligible in λ. We assume that either all policy descriptions p are of equal length, or we require that $\mathcal{A}$ output p_0 and p_1 with $|p_0| = |p_1|$.

Unlinkability could be formalized via a game where an adversary is given two signatures and must decide whether they were created using the same key. Indistinguishability implies unlinkability, as an adversary against the latter could be used to build another one against indistinguishability, who can simulate the unlinkability game by using the received signing keys to produce signatures.

Discussion. The unforgeability and indistinguishability notions we have defined above are basic, intuitive, and suffice for many applications. However, they have some weaknesses, and some applications call for stronger requirements.

First, we claim that indistinguishability does not always provide the privacy we may expect. To see this, consider a policy checker PC such that for every message m there is only one p with $(p, m) \in \mathcal{L}(\mathsf{PC})$. (See our construction of group signatures in Section 5.1 for an example of such a PC.) Now consider a scheme which satisfies indistinguishability, and modify the scheme so that the key contains the policy and the signing algorithm appends the policy to the signature. This scheme clearly does not hide the policy, yet still satisfies indistinguishability. Indeed, in $\mathbf{Exp}^{\mathrm{IND}}_{\mathcal{PBS}}$, in order to satisfy $\mathsf{PC}((p_0, m), w_0) = 1 = \mathsf{PC}((p_1, m), w_1)$, the adversary must return $p_0 = p_1$. If the signatures in the original scheme have not revealed the bit b then attaching the same policy to both will not do so either. (Note however, that a scheme appending sk to a signature would not satisfy the notion.) The notion of simulatability we provide below will fill the gap. It asks that there is a simulator which can create simulated signatures without having access to any

$\underline{\text{proc INITIALIZE}}$ $\boxed{\mathbf{Exp}^{\text{SIM}}_{\mathcal{PBS}}}$
$b \leftarrow\!\!\$\ \{0,1\}$; $j \leftarrow 0$
$(pp_0, msk_0, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
$(pp_1, msk_1) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$; Return (pp_b, msk_b)

$\underline{\text{proc KEY}(p)}$
$j \leftarrow j + 1$
$sk_0 \leftarrow\!\!\$\ \mathsf{SKeyGen}(tr, p)$; $sk_1 \leftarrow\!\!\$\ \mathsf{KeyGen}(msk_1, p)$
$Q[j][1] \leftarrow p$; $Q[j][2] \leftarrow sk_1$; Return sk_b

$\underline{\text{proc SIGNATURE}(i, m, w)}$
If $i \notin [j]$ then return $\perp$
If $\mathsf{PC}((Q[i][1], m), w) = 1$ then $\sigma_0 \leftarrow\!\!\$\ \mathsf{SimSign}(tr, m)$
Else $\sigma_0 \leftarrow \perp$
$\sigma_1 \leftarrow\!\!\$\ \mathsf{Sign}(Q[i][2], m, w)$; Return σ_b

$\underline{\text{proc FINALIZE}(b')}$
Return $(b = b')$

$\underline{\text{proc INITIALIZE}}$ $\boxed{\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS}}}$
$(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
$Q_K \leftarrow \emptyset$; $Q_S \leftarrow \emptyset$; Return pp

$\underline{\text{proc SKEYGEN}(p)}$
$sk \leftarrow\!\!\$\ \mathsf{SKeyGen}(tr, p)$; $Q_K \leftarrow Q_K \cup \{p\}$
Return sk

$\underline{\text{proc SIMSIGN}(m)}$
$\sigma \leftarrow\!\!\$\ \mathsf{SimSign}(tr, m)$; $Q_S \leftarrow Q_S \cup \{(m, \sigma)\}$;
Return σ

$\underline{\text{proc FINALIZE}(m, \sigma)}$
If $\mathsf{Verify}(pp, m, \sigma) = 0$ then return false
If $(m, \sigma) \in Q_S$ then return false
$(p, w) \leftarrow \mathsf{Extr}(tr, m, \sigma)$
If $p \notin Q_K$ or $\mathsf{PC}((p, m), w) = 0$ then return true
Return false

Figure 2: Games defining simulatability and extractability for PBS

signing key or witness, and that these signatures are indistinguishable from real signatures.

With regard to unforgeability, one issue is that in general it cannot be efficiently verified whether an adversary has won the unforgeability game, as this involves checking whether $(p, m) \in \mathcal{L}(\mathsf{PC})$ for all p queried to MAKESK and m from the adversary's final output, and membership in $\mathcal{L}(\mathsf{R})$ may not be efficiently decidable. (This is the case for $\mathcal{L}(\mathsf{R})$ defined in Equation (4) in Section 5.1.) Although not a problem in itself, it can become one, for example when using the notion in a proof by game hopping, as a distinguisher between two games must efficiently determine whether an adversary has won the game. (See Appendix A for a proof that relies on this fact.) The extractability notion we will provide below will fill this gap as well as be more useful in applications. It requires that from a valid signature, using a trapdoor one can extract a policy and a valid witness. To satisfy this notion, a signature must contain information on the policy and can thus not hide its length. For simplicity, we assume from now on that all policies are of the same length.

Simulatability. We formalize *simulatability* by requiring that there exist the following algorithms: $\mathsf{SimSetup}$, which outputs parameters and a master key that are indistinguishable from those output by Setup, as well as a trapdoor; $\mathsf{SKeyGen}$, which outputs keys indistinguishable from those output by KeyGen; and $\mathsf{SimSign}$, which on input the trapdoor and a message (but no signing key nor witness) produces signatures that are indistinguishable from regular signatures.

In particular, with $\mathbf{Exp}^{\text{SIM}}_{\mathcal{PBS}}$ defined in Figure 2, we require that for every PT adversary $\mathcal{A}$ we have $\mathbf{Adv}^{\text{SIM}}_{\mathcal{PBS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\text{SIM}}_{\mathcal{PBS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}] - \frac{1}{2}$ is negligible in λ. Note that in all our constructions, tr contains msk and $\mathsf{SKeyGen}$ is defined as KeyGen. We included $\mathsf{SKeyGen}$ to make the definition more general.

Extractability. We define our notion in the spirit of *"sim-ext" security* for signatures of knowledge [CL06]. Let $\mathbf{Adv}^{\text{EXT}}_{\mathcal{PBS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}]$ with $\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS}}$ defined on the right of Figure 2. We say that $\mathcal{PBS}$ has *extractability* if there exists an algorithm Extr, which taking a trapdoor, a message and a signature outputs a pair $(p, w) \in \{0,1\}^*$, such that $\mathbf{Adv}^{\text{EXT}}_{\mathcal{PBS},\mathcal{A}}(\cdot)$ is negligible for every PT $\mathcal{A}$.

Although the definition might not seem completely intuitive at first, it implies that, as long as the adversary outputs a valid message/signature pair and does not simply copy a SIMSIGN query/response pair, the only signed messages it can output are those that satisfy the policy of one of the queried keys: assume $\mathcal{A}$ outputs (m^*, σ^*) such that $(*)$ for all $p \in Q_K$: $(p, m^*) \notin \mathcal{L}(\mathsf{PC})$. Then let $(p^*, w^*) \leftarrow \mathsf{Extr}(tr, m, \sigma)$. If $\mathsf{PC}((p^*, m^*), w^*) = 0$, the adversary wins $\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS}}$. On the other hand, if $\mathsf{PC}((p^*, m^*), w^*) = 1$ then $(p^*, m^*) \in \mathcal{L}(\mathsf{PC})$, thus by $(*)$ we have $p^* \notin Q_K$ and it wins too. Note that this notion corresponds to *strong unforgeability* for signature schemes.

Sim-ext security implies indistinguishability and unforgeability. In Appendix A we show that our two latter security notions are indeed strengthenings of the former two:

Theorem 1. *Any policy-based signature scheme which satisfies simulatability satisfies indistinguishability. Any PBS scheme which satisfies simulatability and extractability satisfies unforgeability.*

4 Constructions of Policy-Based Signature Schemes

We first show that PBS satisfying SIM+EXT can be achieved for any language in **NP**. Then we develop more efficient schemes for specific policy languages that are useful in applications.

4.1 Generic Constructions

We now show how to construct policy-based signatures satisfying simulatability and extractability (and, by Theorem 1, indistinguishability and unforgeability) for any **NP**-relation PC. Section 5 shows that the assumptions we make are not only sufficient but necessary.

A first approach could be the following, similar to the generic construction of group signatures in [BMW03]: The issuer creates a signature key pair (mvk, msk) and publishes mvk as pp. When a user is issued a key for a policy p, the issuer creates a key pair (vk_U, sk_U), signs $p \| vk_U$ and sends this certificate to the user together with (p, vk_U, sk_U). Now in order to sign a message m, the user first signs it under sk_U, thereby establishing a chain $mvk \to vk_U \to m$ via the certificate and the signature. In order to remain anonymous, the actual signature is a (zero-knowledge) proof of knowledge of such a chain and the fact that the message satisfies the policy signed in the certificate.

While this approach yields a scheme satisfying indistinguishability and unforgeability, it would fail to achieve extractability. We thus choose a different approach: The user's key is simply a signature from the issuer on the policy. Now to sign a message, the user first picks a key pair (ovk, osk) for a strongly unforgeable one-time signature scheme[1] and makes a zero-knowledge proof π that he knows either (I) an issuer signature on a policy p such that $(p, m) \in \mathcal{L}(\mathsf{PC})$ or (II) an issuer signature on ovk. Finally, he adds a signature under ovk of both the message and the proof. As we will see, this construction satisfies both SIM (where the simulator can make a signature on ovk and use clause (II) for the proof) and EXT (as π is a proof of knowledge).

We formalize the above: Let $\mathcal{S}ig = (\mathsf{KeyGen}_{\mathrm{sig}}, \mathsf{Sign}_{\mathrm{sig}}, \mathsf{Verify}_{\mathrm{sig}})$ be a signature scheme which is unforgeable under chosen-message attacks (UF-CMA), $\mathcal{O}t\mathcal{S}ig = (\mathsf{KeyGen}_{\mathrm{ots}}, \mathsf{Sign}_{\mathrm{ots}}, \mathsf{Verify}_{\mathrm{ots}})$ a strongly unforgeable one-time signature scheme and let $\mathcal{PKE} = (\mathsf{KeyGen}_{\mathrm{pke}}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA-secure public-key encryption scheme. For a policy checker PC we define the following **NP**-relation:

$$\begin{aligned}\big((pk, mvk, C_p, C_s, C_w, ovk, m), (p, s, w, \rho_p, \rho_s, \rho_w)\big) \;&\in\; R_{\mathrm{NP}} \\ \iff\; C_p = \mathsf{Enc}(pk, p; \rho_p) \;&\wedge\; C_s = \mathsf{Enc}(pk, s; \rho_s) \;\wedge\; C_w = \mathsf{Enc}(pk, w; \rho_w) \;\wedge \\ &\big[\big(\mathsf{Verify}_{\mathrm{sig}}(mvk, 1\|p, s) = 1 \;\wedge\; \mathsf{PC}((p, m), w) = 1\big) \;\vee\; \mathsf{Verify}_{\mathrm{sig}}(mvk, 0\|ovk, s) = 1\big]\end{aligned} \tag{1}$$

Let $\mathcal{NIZK} = (\mathsf{Setup}_{\mathrm{nizk}}, \mathsf{Prove}, \mathsf{Verify}_{\mathrm{nizk}})$ be a non-interactive zero-knowledge (NIZK) proof system for $\mathcal{L}(R_{\mathrm{NP}})$. Our construction $\mathcal{PBS}$ for a policy checker PC is detailed in Figure 3, and in Appendix B we prove the following:

Theorem 2. *If $\mathcal{PKE}$ satisfies IND-CPA, $\mathcal{S}ig$ is UF-CMA, $\mathcal{O}t\mathcal{S}ig$ is a strongly unforgeable one-time signature scheme and $\mathcal{NIZK}$ is a NIZK proof system for $\mathcal{L}(R_{NP})$ then $\mathcal{PBS}[\mathcal{PKE}, \mathcal{S}ig, \mathcal{O}t\mathcal{S}ig, \mathcal{NIZK}]$, defined in Figure 3, satisfies simulatability and extractability.*

We now present a much simpler construction of PBS by relying on a more advanced cryptographic primitive: simulation-extractable (SE) NIZK proofs [Gro06] (see Appendix F for the definition). Let $\mathcal{S}ig = (\mathsf{KeyGen}_{\mathrm{sig}},$

[1] In such a scheme it must be infeasible for an adversary, after receiving a verification key ovk and after obtaining a signature σ on one message m of his choice, to output a signature σ^* on a message m^*, such that $(m, \sigma) \neq (m^*, \sigma^*)$.

$\underline{\mathsf{Setup}(1^\lambda)}$
$crs \leftarrow\!\!\$\ \mathsf{Setup}_{\text{nizk}}(1^\lambda)$
$(pk, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pke}}(1^\lambda)$
$(mvk, msk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$
Return $pp \leftarrow (crs, pk, mvk)$ and msk

$\underline{\mathsf{KeyGen}(msk, p)}$
$s \leftarrow\!\!\$\ \mathsf{Sign}_{\text{sig}}(msk, 1\|p)$
Return $sk_p \leftarrow (pp, p, s)$

$\underline{\mathsf{Sign}(sk_p, m, w)}$
Parse $((crs, pk, mvk), p, s) \leftarrow sk_p$
If $\mathsf{PC}((p, m), w) = 0$ then return $\perp$
$(ovk, osk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{ots}}(1^\lambda)$
$\rho_p, \rho_s, \rho_w \leftarrow\!\!\$\ \{0,1\}^\lambda$; $C_p \leftarrow \mathsf{Enc}(pk, p; \rho_p)$
$C_s \leftarrow \mathsf{Enc}(pk, s; \rho_s)$; $C_w \leftarrow \mathsf{Enc}(pk, w; \rho_w)$
$\pi \leftarrow\!\!\$\ \mathsf{Prove}(crs, (pk, mvk, C_p, C_s, C_w, ovk, m), (p, s, w, \rho_p, \rho_s, \rho_w))$
$\tau \leftarrow\!\!\$\ \mathsf{Sign}_{\text{ots}}(osk, (m, C_p, C_s, C_w, \pi))$
Return $\sigma \leftarrow (ovk, C_p, C_s, C_w, \pi, \tau)$

$\underline{\mathsf{Verify}(pp, m, \sigma)}$
Parse $(crs, pk, mvk) \leftarrow pp$; $(ovk, C_p, C_s, C_w, \pi, \tau) \leftarrow \sigma$
Return 1 iff
$\mathsf{Verify}_{\text{nizk}}(crs, (pk, mvk, C_p, C_s, C_w, ovk, m), \pi) = 1$
and $\mathsf{Verify}_{\text{ots}}(ovk, (m, C_p, C_s, C_w, \pi), \tau) = 1$

$\underline{\mathsf{SimSetup}(1^\lambda)}$
$crs \leftarrow\!\!\$\ \mathsf{Setup}_{\text{nizk}}(1^\lambda)$
$(pk, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pke}}(1^\lambda)$
$(mvk, msk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$
Return $pp \leftarrow (crs, pk, mvk)$, msk and $tr \leftarrow (msk, dk)$

$\underline{\mathsf{SKeyGen}((msk, dk), p)}$
$s \leftarrow\!\!\$\ \mathsf{Sign}_{\text{sig}}(msk, 1\|p)$
Return $sk_p \leftarrow (pp, p, s)$

$\underline{\mathsf{SimSign}((msk, dk), m)}$
$(ovk, osk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{ots}}(1^\lambda)$
$s \leftarrow\!\!\$\ \mathsf{Sign}_{\text{sig}}(msk, 0\|ovk)$
$\rho_p, \rho_s, \rho_w \leftarrow\!\!\$\ \{0,1\}^\lambda$; $C_p \leftarrow \mathsf{Enc}(pk, 0; \rho_p)$
$C_s \leftarrow \mathsf{Enc}(pk, s; \rho_s)$; $C_w \leftarrow \mathsf{Enc}(pk, 0; \rho_w)$
$\pi \leftarrow\!\!\$\ \mathsf{Prove}(crs, (pk, mvk, C_p, C_s, C_w, ovk, m), (0, s, 0, \rho_p, \rho_s, \rho_w))$
$\tau \leftarrow\!\!\$\ \mathsf{Sign}_{\text{ots}}(osk, (m, C_p, C_s, C_w, \pi))$
Return $\sigma \leftarrow (ovk, C_p, C_s, C_w, \pi, \tau)$

$\underline{\mathsf{Extr}((msk, dk), m, \sigma)}$
Parse $(ovk, C_p, C_s, C_w, \pi, \tau) \leftarrow \sigma$
$p \leftarrow \mathsf{Dec}(dk, C_p)$; $w \leftarrow \mathsf{Dec}(dk, C_w)$
Return (p, w)

Figure 3: Generic construction of PBS

$\mathsf{Sign}_{\text{sig}}, \mathsf{Verify}_{\text{sig}}$) be a signature scheme and for a policy checker PC let $\mathcal{NIZK} = (\mathsf{Setup}_{\text{nizk}}, \mathsf{Prove}, \mathsf{Verify}_{\text{nizk}}, \mathsf{SimSetup}_{\text{nizk}}, \mathsf{SimProve}, \mathsf{Extr}_{\text{nizk}})$ be a SE-NIZK for the following **NP**-relation, whose statements are of the form $X = (vk, m)$ with witnesses $W = (p, c, w)$ and

$$((vk, m), (p, c, w)) \in R_{\text{NP}} \iff \mathsf{Verify}_{\text{sig}}(vk, p, c) = 1 \ \wedge\ ((p, m), w) \in \mathsf{PC}$$

Then the scheme in Figure 4 is a PBS for PC which satisfies simulatability and extractability. This follows from Theorem 4 in Section 6, which is about a generalization of the scheme in Figure 4.

4.2 Efficient Construction via Groth-Sahai Proofs

Our efficient construction of PBS will be defined over a *bilinear group*. This is a tuple $(p, \mathbb{G}, \mathbb{H}, \mathbb{T}, G, H)$, where $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{T}$ are groups of prime order p, generated by G and H, respectively, and $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{T}$ is a bilinear map such that $e(G, H)$ generates $\mathbb{T}$. We denote the group operation multiplicatively and let $1_{\mathbb{G}}$, $1_{\mathbb{H}}$ and $1_{\mathbb{T}}$ denote the neutral elements of $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{T}$. Groth-Sahai proofs [GS08] let us prove that there exists a set of elements $(\mathbf{X}, \mathbf{Y}) = (X_1, \ldots, X_n, Y_1, \ldots, Y_\ell) \in \mathbb{G}^n \times \mathbb{H}^\ell$ which satisfy equations $\mathsf{E}(\underline{\mathbf{X}}, \underline{\mathbf{Y}})$ of the form

$$\prod_{i=1}^{k} e(P_i, Q_i) \prod_{j=1}^{\ell} e(A_j, \underline{Y_j}) \prod_{i=1}^{n} e(\underline{X_i}, B_i) \prod_{i=1}^{n} \prod_{j=1}^{\ell} e(\underline{X_i}, \underline{Y_j})^{\gamma_{ij}} = 1_{\mathbb{T}} \tag{2}$$

Such an equation E is called a *pairing-product equation*[2] (PPE) and is uniquely defined by its constants $\mathbf{P}, \mathbf{Q}, \mathbf{A}, \mathbf{B}$ and $\Gamma := (\gamma_{ij})_{i \in [n], j \in [\ell]}$. These equations have already found many uses in cryptography, of which the following two are relevant in our context: they can define the verification predicate of a digital signature (see [AFG+10] and the discussion therein), or witness the fact that a ciphertext encrypts a certain

[2]This is a *simulatable* pairing-product equation, that is, one for which Groth-Sahai proofs can be made zero-knowledge.

$\underline{\mathsf{Setup}(1^\lambda)}$
$crs \leftarrow\!\!\$\ \mathsf{Setup}_{\text{nizk}}(1^\lambda)$
$(mvk, msk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$
Return $pp \leftarrow (crs, mvk)$, msk

$\underline{\mathsf{KeyGen}(msk, p)}$
$c \leftarrow\!\!\$\ \mathsf{Sign}_{\text{sig}}(msk, p)$
Return $sk \leftarrow (pp, p, c)$

$\underline{\mathsf{Sign}(sk = ((crs, mvk), p, c), m, w)}$
Return $\sigma \leftarrow\!\!\$\ \mathsf{Prove}(crs, (mvk, m), (p, c, w))$

$\underline{\mathsf{Verify}(pp = (crs, mvk), m, \sigma)}$
Return $\mathsf{Verify}_{\text{nizk}}(crs, (mvk, m), \sigma)$

$\underline{\mathsf{SimSetup}(1^\lambda)}$
$(crs, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}_{\text{nizk}}(1^\lambda)$
$(mvk, msk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$
Return $pp \leftarrow (crs, mvk)$, msk, $tr_{\text{pbs}} \leftarrow (pp, msk, tr)$

$\underline{\mathsf{SKeyGen}((pp, msk, tr), p)}$
$c \leftarrow\!\!\$\ \mathsf{Sign}_{\text{sig}}(msk, p)$
Return $sk \leftarrow (pp, p, c)$

$\underline{\mathsf{SimSign}(((crs, mvk), msk, tr), m)}$
Return $\sigma \leftarrow\!\!\$\ \mathsf{SimProve}(crs, tr, (mvk, m))$

$\underline{\mathsf{Extr}(((crs, mvk), msk, tr), m, \sigma)}$
$(p, c, w) \leftarrow \mathsf{Extr}_{\text{nizk}}(tr, (mvk, m), \sigma)$; Return (p, w)

Figure 4: PBS based on SE-NIZKs.

value (see Appendix F.3). Our aim is to construct policy-based signatures where policies define (sets of) PPEs, which must be satisfied by the message and the witness.

Groth and Sahai define a setup algorithm which on input a bilinear group outputs a common reference string crs and an extraction key xk. On input crs, an equation E and a satisfying witness $(\mathbf{X}, \mathbf{Y})$, algorithm $\mathsf{Prove}_{\text{gs}}$ outputs a proof π. Proofs are verified by $\mathsf{Verify}_{\text{gs}}(crs, \mathsf{E}(\cdot, \cdot), \pi)$. Under the SXDH assumption (see [GS08] for the definition), Groth-Sahai (GS) proofs are *witness-indistinguishable* [FS90], that is, proofs for an equation using different witnesses are computationally indistinguishable. Moreover, they are *extractable* and thus proofs of knowledge [DMP87]: From every valid proof π, $\mathsf{Extr}_{\text{gs}}(xk, \mathsf{E}(\cdot, \cdot), \pi)$ extracts a witness $(\mathbf{X}, \mathbf{Y})$ such that $\mathsf{E}(\mathbf{X}, \mathbf{Y}) = 1$.

In our Groth-Sahai-based construction of PBS, messages and witnesses will be group elements and a policy defines a set of equations as in (2) that have to be satisfied. The policy checker is thus defined as follows: the policy p defines a set of equations $(\mathsf{E}_1, \ldots, \mathsf{E}_n)$ and $\mathsf{PC}((p, m), w) = 1$ iff $\mathsf{E}_i(m, w) = 1$ for all $i \in [n]$, where $m \in \mathbb{G}^{n_m} \times \mathbb{H}^{\ell_m}$ and $w \in \mathbb{G}^{n_w} \times \mathbb{H}^{\ell_w}$.

GS proofs only allow us to extract group elements; however, an equation—and thus a policy—is defined by a set of group elements and exponents γ_{ij}. In order to hide a policy, we need to swap the roles of constants and variables in an equation, as this will enable us to hide the policy defined by the constants. We first transform equations as in (2) into a set of equivalent equations without exponents. To do so, we introduce auxiliary variables $\widehat{Y}_{ij}$, add $i \cdot j$ new equations and define the set $\mathsf{E}^{(\text{no-exp})}$ as follows:

$$\prod e(P_i, Q_i) \prod e(A_j, \underline{Y_j}) \prod e(\underline{X_i}, B_i) \prod\prod e(\underline{X_i}, \underline{\widehat{Y}_{ij}}) = 1_{\mathbb{T}} \quad \wedge \quad \bigwedge\nolimits_{i,j} e(G, \underline{\widehat{Y}_{ij}}) = e(G^{\gamma_{ij}}, \underline{Y_j}) \qquad (3)$$

A witness $(\mathbf{X}, \mathbf{Y})$ satisfies E in (2) iff $(\mathbf{X}, \mathbf{Y}, (\widehat{Y}_{ij} := Y_j^{\gamma_{ij}})_{i,j})$ satisfies the set of equations $\mathsf{E}^{(\text{no-exp})}$ in (3). Now we can show that a (clear) message $(\mathbf{M}, \mathbf{N})$ satisfies a "hidden" policy defined by equation E, witnessed by elements $(\mathbf{V}, \mathbf{W})$, since we can express policies as sets of group elements.

Our second building block are structure-preserving signatures [AFG+10], which were designed to be combined with GS proofs: their keys, messages and signatures consist only of elements from $\mathbb{G}$ and $\mathbb{H}$ and signatures are verified by evaluating PPEs. GS proofs let us prove knowledge of keys, messages, and/or signatures such that these values satisfy signature verification, without revealing anything beyond this fact.

Our construction now follows the blueprint of the generic scheme in Figure 3. The setup creates a CRS for GS proofs and a key pair (mvk, msk) for a structure-preserving scheme $\mathcal{S}ig_{\text{sp}}$. (Note that here we need not encrypt any witnesses like in the generic construction, since GS proofs are extractable.) We transform every PPE E contained in a policy to a set of equations $\mathsf{E}^{(\text{no-exp})}$ without exponents. The policies can thus be expressed as sets of group elements describing the equations $\mathsf{E}^{(\text{no-exp})}$, which can be signed by $\mathcal{S}ig_{\text{sp}}$.

As for the generic construction, a signing key is a signature on the policy under msk and signing is done by choosing a one-time signature key pair (ovk, osk), proving a statement analogous to (1) and signing the proof and the message with osk. A further technical obstacle is that we need to express the disjunction

in the statement to be proven as (a conjunction of) sets of PPEs. We achieve this by following Groth's approach in [Gro06]. The details of the construction can be found in Appendix C.

A simple use case. Messages that are elements of bilinear groups and policies demanding that they satisfy PPEs will prove useful to construct other cryptographic schemes like group signatures. Yet, our pairing-based construction might seem too abstract for deploying PBS to manage signing rights in a company—one of the motivations given in the introduction.

However, consider the following simple example: A company issues keys to their employees which should allow them to sign only messages $h\|m$ that start with a particular *header* h. (E.g. h could be "Contract with company X", so employees are limited to signing contracts with X.) This can be implemented by mapping messages $h\|m$ to $(F(h), F(m))$ via a collision-resistant hash function $F\colon \{0,1\}^* \to \mathbb{G}$. (E.g. first hash to $\mathbb{Z}_p$ via some f and then set $F(x) = G^{f(x)}$.) The policy p^* requiring messages to start with h^* can then be expressed as $\mathsf{PC}((p^*, h\|m)) = 1 \Leftrightarrow e(F(h^*), H)\, e(F(h), H^{-1}) = 1$.

Another possibility would be to additionally demand that an employee hold a credential, of which the required type could even depend on h^*, which she must use as a witness when signing. (There are numerous constructions of credentials (digital signatures) which are verified via PPEs; see e.g. [AFG+10].)

5 Applications and Implications

Here we illustrate how PBSs can provide a unifying framework for work on advanced forms of signatures and beyond, capturing some primitives as special cases and allowing others to be derived in simple and natural ways. We begin in Section 5.1 by showing how PBSs allow one to easily obtain group signatures [BMW03]. In Section 5.2 we show that they imply signatures of knowledge [CL06] and in Appendix D how PBSs easily yield attribute-based signatures [MPR11]. These applications are illustrative rather than exhaustive.

Section 4.1 shows which primitives are sufficient for policy-based signatures. We now ask the converse question, namely which primitives are necessary, that is, which fundamental cryptographic primitives are implied by PBS? In Section 5.2 we show that PBSs imply simulation-extractable NIZKs and IND-CPA encryption. As a result [Sah99] they imply IND-CCA public-key encryption. This means that the sufficient assumptions we make in our constructions of Section 4.1 are also necessary.

5.1 CCA-Secure Group Signatures from Policy-Based Signatures

Group signatures [Cv91] let members sign anonymously on behalf of a group. To deter misuse, the group manager holds a secret key which can *open* signatures, that is, reveal the member that made the signature.

The BMW model. As defined in [BMW03], a group-signature scheme $\mathcal{GS} = (\mathsf{GKg}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open})$ is a 4-tuple of PT algorithms. On input the security parameter 1^λ and the group size 1^n, group-key generation algorithm GKg returns the group public key gpk, the manager's secret key $gmsk$ and a vector of member secret keys $\boldsymbol{gsk}$. On input $\boldsymbol{gsk}[i]$ and a message $m \in \{0,1\}^*$, group signing algorithm GSig returns a group signature γ by member i on m. On input gpk, m and γ, verification algorithm GVf outputs a bit. On input $gmsk, m$ and γ, the opening algorithm Open returns an identity $i \in [n]$ or $\perp$.

We say that $\mathcal{GS}$ is correct if for all $\lambda, n \in \mathbb{N}$, all $(gpk, gmsk, \boldsymbol{gsk}) \in [\mathsf{GKg}(1^\lambda, 1^n)]$, all $i \in [n]$, all $m \in \{0,1\}^*$ and all $\gamma \in [\mathsf{GSig}(\boldsymbol{gsk}[i], m)]$, we have $\mathsf{GVf}(gpk, m, \gamma) = 1$ and $\mathsf{Open}(gmsk, m, \gamma) = i$. Security for $\mathcal{GS}$ is defined via the experiments $\mathbf{Exp}^{\text{ANON}}_{\mathcal{GS}}$ and $\mathbf{Exp}^{\text{TRC}}_{\mathcal{GS}}$ given in Figure 5. Following [BMW03], in $\mathbf{Exp}^{\text{ANON}}_{\mathcal{GS}}$ we allow the adversary only one call to his LR oracle. We say that $\mathcal{GS}$ is *fully anonymous* if $\mathbf{Adv}^{\text{ANON}}_{\mathcal{GS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\text{ANON}}_{\mathcal{GS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}] - \frac{1}{2}$ is negligible in λ for all PT adversaries $\mathcal{A}$. We say that $\mathcal{GS}$ is *traceable* if $\mathbf{Adv}^{\text{TRC}}_{\mathcal{GS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\text{TRC}}_{\mathcal{GS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}]$ is negligible in λ for all PT adversaries $\mathcal{A}$.

Construction. We show how to construct group signatures from CCA-secure public-key encryption and policy-based signatures. Since the former can be constructed from the latter (as we show in Section 5.2), this means that PBS implies group signatures. The main idea is to define a group signature as a ciphertext plus a PBS. When making a group signature on a message m, a member is required to encrypt her identity

$\mathbf{Exp}^{\text{ANON}}_{\mathcal{GS}}$

proc INITIALIZE
$(gpk, gmsk, \boldsymbol{gsk}) \leftarrow\!\!\$\ \mathsf{GKg}(1^k, 1^n)$
$b \leftarrow\!\!\$\ \{0,1\}$; $Q \leftarrow \emptyset$
Return $(gpk, \boldsymbol{gsk})$

proc LR(i_0, i_1, m)
$\gamma \leftarrow\!\!\$\ \mathsf{GSig}(\boldsymbol{gsk}[i_b], m)$
$Q \leftarrow Q \cup \{\gamma\}$; return γ

proc OPEN(m, γ)
If $\gamma \in Q$ then return $\bot$
Return $\mathsf{Open}(gmsk, m, \gamma)$

proc FINALIZE(b')
Return $(b = b')$

$\mathbf{Exp}^{\text{TRC}}_{\mathcal{GS}}$

proc INITIALIZE
$(gpk, gmsk, \boldsymbol{gsk}) \leftarrow\!\!\$\ \mathsf{GKg}(1^k, 1^n)$
$Q_C \leftarrow \emptyset$; $Q_S \leftarrow \emptyset$; return $(gpk, gmsk)$

proc CORRUPT(i)
$Q_C \leftarrow Q_C \cup \{i\}$; return $\boldsymbol{gsk}[i]$

proc GSIG(i, m)
$Q_S \leftarrow Q_S \cup \{(i, m)\}$; return $\mathsf{GSig}(\boldsymbol{gsk}[i], m)$

proc FINALIZE(m, γ)
If $\mathsf{GVf}(gpk, m, \gamma) = 0$ then return false
If $\mathsf{Open}(gmsk, m, \gamma) = \bot$ then return true
$i \leftarrow \mathsf{Open}(gmsk, m, \gamma)$
If $i \in [n]$, $i \notin Q_C$ and $(i, m) \notin Q_S$ then return true
Return false

Figure 5: Games defining full anonymity and traceability for group signatures

as c and then sign (c, m). This is enforced by issuing to the member a PBS key whose policy ensures that c must be an encryption of the member's identity.

Let $\mathcal{PKE} = (\mathsf{KeyGen}_{\text{pke}}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme satisfying IND-CCA and let $\mathcal{PBS} = (\mathsf{Setup}, \mathsf{KeyGen}_{\text{pbs}}, \mathsf{Sign}, \mathsf{Verify})$ be a PBS for the following **NP**-relation:

$$\mathsf{PC}\big(((ek, i), (c, m)), r\big) \iff c = \mathsf{Enc}(ek, i; r) \ . \tag{4}$$

Our group-signature scheme $\mathcal{GS}[\mathcal{PKE}, \mathcal{PBS}]$ is defined as follows:

$\underline{\mathsf{GKg}(1^\lambda, 1^n)}$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pke}}(1^\lambda)$
For $i = 1, \ldots, n$ do
$sk_i \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pbs}}(msk, (ek, i))$
$\boldsymbol{gsk}[i] \leftarrow (pp, ek, i, sk_i)$
Return $(gpk \leftarrow (pp, ek), gmsk \leftarrow dk, \boldsymbol{gsk})$

$\underline{\mathsf{GSig}((pp, ek, i, sk_i), m)}$
$r \leftarrow\!\!\$\ \{0,1\}^\lambda$
$c \leftarrow \mathsf{Enc}(ek, i; r)$
$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk_i, (c, m), r)$
Return (c, σ)

$\underline{\mathsf{GVf}((pp, ek), m, (c, \sigma))}$
Return $\mathsf{Verify}(pp, (c, m), \sigma)$

$\underline{\mathsf{Open}(gmsk, m, (c, \sigma))}$
If $\mathsf{Verify}(pp, (c, m), \sigma) = 0$
Then return $\bot$
Return $\mathsf{Dec}(gmsk, c)$

Theorem 3. *If $\mathcal{PKE}$ is an IND-CCA-secure public-key encryption scheme and $\mathcal{PBS}$ is a policy-based signature scheme for PC defined in (4) satisfying simulatability and extractability then $\mathcal{GS}[\mathcal{PKE}, \mathcal{PBS}]$ is a group-signature scheme satisfying full anonymity and traceability.*

The proof can be found in Appendix E. In Appendix F.3 we give an encryption scheme such that (4) lies in the language of our efficient PBS from Section 4.2.

5.2 Other Primitives Implied by PBS

Section 4.1 shows which primitives are sufficient for policy-based signatures. We now show which primitives are necessary, that is, which fundamental cryptographic primitives are implied by PBS.

Simulation-extractable NIZK proofs. Groth [Gro06] introduced a notion of simulation-soundness for NIZK proofs of knowledge. It requires that even provided with an oracle for simulated proofs, an adversary cannot produce a new proof from which the extractor fails to extract a witness. (See Appendix F.1 for a definition.) Let R be an **NP**-relation and let PC and p^* be such that $\mathsf{PC}((p^*, x), w) = \mathsf{R}(x, w)$. Let $\mathcal{PBS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SimSetup}, \mathsf{SKeyGen}, \mathsf{SimSign}, \mathsf{Extr})$ be a PBS for PC which satisfies simulatability and extractability. Then the following is a simulation-extractable NIZK proof system:

$\underline{\mathsf{Setup}_{\mathrm{nizk}}(1^\lambda)}$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$sk \leftarrow \mathsf{KeyGen}(msk, p^*)$
Return $crs \leftarrow (pp, sk)$

$\underline{\mathsf{Prove}((pp, sk), x, w)}$
Return $\pi \leftarrow \mathsf{Sign}(sk, x, w)$

$\underline{\mathsf{Verify}_{\mathrm{nizk}}((pp, sk), x, \pi)}$
Return $\mathsf{Verify}(pp, x, \pi)$

$\underline{\mathsf{Extr}_{\mathrm{nizk}}(tr, x, \pi)}$
$(p, w) \leftarrow \mathsf{Extr}(tr, x, \pi)$
Return w

$\underline{\mathsf{SimSetup}_{\mathrm{nizk}}(1^\lambda)}$
$(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
$sk \leftarrow \mathsf{SKeyGen}(tr, p^*)$
Return $crs \leftarrow (pp, sk)$ and tr

$\underline{\mathsf{SimProve}((pp, sk), tr, x)}$
Return $\pi \leftarrow \mathsf{SimSign}(tr, x)$

Showing that SIM of $\mathcal{PBS}$ implies zero-knowledge of $\mathcal{NIZK}$ and that EXT of $\mathcal{PBS}$ implies simulation-extractability of $\mathcal{NIZK}$ is straightforward.

Public-key encryption. Interpreting policies as plaintexts hidden in a signature, which can be "decrypted" using Extr, PBS even imply public-key encryption: Let $\mathcal{PBS}$ be for PC s.t. $\mathsf{PC}((p, m^*), w^*) = 1$ for some m^*, w^* and all p. Then the following as an IND-CPA secure PKE:

$\underline{\mathsf{KeyGen}_{\mathrm{pke}}(1^\lambda)}$
$(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
Return $(pk \leftarrow (pp, msk),\ dk \leftarrow tr)$

$\underline{\mathsf{Enc}((pp, msk), x)}$
$sk \leftarrow\!\!\$\ \mathsf{KeyGen}(msk, x)$
Return $c \leftarrow\!\!\$\ \mathsf{Sign}(sk, m^*, w^*)$

$\underline{\mathsf{Dec}(dk, c)}$
$(x, w) \leftarrow \mathsf{Extr}(dk, m^*, c)$
Return x

Using the results by Sahai [Sah99], the above NIZK and PKE can be combined to construct a CCA-secure PKE from PBS (see Appendix F.2 for more details).

Signatures of knowledge. In signatures of knowledge (SoK) [CL06] there is a setup outputting trusted parameters, but no authority. Anyone can make a signature on a message m w.r.t. an **NP**-relation R and a statement x, if they know a witness w for x, i.e., $R(x, w) = 1$. Security is defined by simulatability and extractability notions analogous to ours for PBS. We refer to [CL06] for the definitions.

To construct a SoK, we use a PBS which signs messages of the form (R, x, m) and require a signer to know a witness w for x w.r.t. R. In particular, let $\mathcal{PBS}$ be a PBS scheme for the policy checker PC s.t. for some policy p^* we have: $\mathsf{PC}((p^*, (R, x, m), w) = 1$ iff $R(x, w) = 1$. Then we define:

$\underline{\mathsf{Setup}_{\mathrm{sok}}(1^\lambda)}$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}_{\mathrm{pbs}}(1^\lambda)$
$sk \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{pbs}}(msk, p^*)$
Return $pp_{\mathrm{sok}} = (pp, sk)$

$\underline{\mathsf{Sign}_{\mathrm{sok}}((pp, sk), R, x, m, w)}$
$c \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{pbs}}(sk, (R, x, m), w)$
Return c

$\underline{\mathsf{Verify}_{\mathrm{sok}}((pp, sk), R, x, m, \sigma)}$
Return $\mathsf{Verify}_{\mathrm{pbs}}(pp, (R, x, m), \sigma)$

There are 3 more algorithms required for SoK: $\mathsf{SimSetup}_{\mathrm{sok}}$ and $\mathsf{SimSign}_{\mathrm{sok}}$ are defined by replacing the respective PBS algorithms by their simulated variants. Finally, $\mathsf{Extr}_{\mathrm{sok}}$ runs $\mathsf{Extr}_{\mathrm{pbs}}$ to get (p, w) and returns w. It is then straightforward to show that this scheme satisfies simulatability and extractability.

6 Delegatable Policy-Based Signatures

In an organization, policies may be hierarchical, reflecting the organization structure. Thus, a president may declare a high-level policy to vice presidents and issue keys to them. Each of the vice presidents augments the policy with their own sub-policies for managers below them, and so on. To support this, we extend PBS to allow delegation. We define and achieve *delegatable* policy-based signatures (DPBS), where a user holding a key for some policy can delegate her key to another user and possibly restrict the associated policy. We formalize this by associating keys to *vectors* of policies and require that keys can (only) sign messages which are allowed under *all* policies associated to the key. In order to restrict the policy at delegation, users can add policies to the associated vector. For ease of notation we generalize PC to policy vectors and define $\mathsf{PC}\big(((p_1, \ldots, p_n), m), (w_1, \ldots, w_n)\big) = 1 \Leftrightarrow \forall i \in [n] : \mathsf{PC}((p_i, m), w_i) = 1$. Thus, a message m satisfies a vector $(p_1, \ldots, p_n)$ iff it satisfies all policies in it, i.e., $((p_1, \ldots, p_n), m) \in \mathcal{L}(\mathsf{PC})$ iff $(p_i, m) \in \mathcal{L}(\mathsf{PC})$ for all $i \in [n]$.

$\mathbf{Exp}^{\text{SIM}}_{\mathcal{DPBS}}$

proc INITIALIZE
$b \leftarrow\!\!\$\ \{0,1\}$
$(pp_0, msk_0, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
$(pp_1, msk_1) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
Return (pp_b, msk_b)

proc SIGNATURE$(sk, \mathbf{p}, m, \mathbf{w})$
If $\mathsf{ChkKey}(pp, sk, \mathbf{p}) = 0$
  then return $\bot$
If $\mathsf{PC}((\mathbf{p}, m), \mathbf{w}) = 1$ then
  $\sigma_0 \leftarrow\!\!\$\ \mathsf{SimSign}(tr, m)$
Else $\sigma_0 \leftarrow \bot$
$\sigma_1 \leftarrow\!\!\$\ \mathsf{Sign}(sk, m, (w_1, \ldots, w_n))$
Return σ_b

proc FINALIZE(b')
Return $(b = b')$

$\mathbf{Exp}^{\text{EXT}}_{\mathcal{DPBS}}$

proc INITIALIZE
$(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
$Q[0] \leftarrow (\varepsilon, msk)$; $Q_S, Q_P \leftarrow \emptyset$; $j \leftarrow 0$;
Return pp

proc DELEGATE(i, p')
If $i \notin \{0, \ldots, j\}$ or $Q[i][2] = \bot$ then return $\bot$
$j \leftarrow j + 1$; $Q[j][1] \leftarrow Q[i][1] \,\|\, p'$
$Q[j][2] \leftarrow\!\!\$\ \mathsf{Delegate}(Q[i][2], p')$

proc REVEALSK(i)
If $i \notin [j]$ then return $\bot$
$sk \leftarrow Q[i][2]$; $Q_P \leftarrow Q_P \cup \{Q[i][1]\}$; return sk

proc SIMSIGN(m)
$\sigma \leftarrow\!\!\$\ \mathsf{SimSign}(tr, m)$; $Q_S \leftarrow Q_S \cup \{(m, \sigma)\}$; return σ

proc FINALIZE(m, σ)
If $\mathsf{Verify}(pp, m, \sigma) = 0$ or $(m, \sigma) \in Q_S$ then return false
$(\mathbf{p}, \mathbf{w}) \leftarrow \mathsf{Extr}(tr, m, \sigma)$
If no $\mathbf{q} \in Q_P$ is a prefix of $\mathbf{p}$ then return true
If $\mathsf{PC}((\mathbf{p}, m), \mathbf{w}) = 0$ then return true
Return false

Figure 6: Games defining simulatability and extractability for delegatable PBS

Consider the following simple use case: A company issues a key to a manager Alice which enables her to sign contracts with companies X, Y and Z. Now Bob is negotiating a contract with Z on behalf of Alice, so she gives Bob a key that only lets him sign contracts with Z.

Delegatable policy-based signatures. Formally, a DPBS scheme consists of the same algorithms as a PBS scheme, except that KeyGen is replaced by a new algorithm $\mathsf{Delegate}$, which takes as input a signing key sk_p for a policy vector $\mathbf{p} = (p_1, \ldots, p_n)$ and a policy p'. It outputs a signing key $sk_{p\|p'}$ allowing to sign a message m if it satisfies the policies $p_1, \ldots, p_n$ and p', i.e., $(p, m) \in \mathcal{L}(\mathsf{PC})$ for all $p \in \{p_1, \ldots, p_n, p'\}$. When the authority issues a signing key, it runs $\mathsf{Delegate}$ with msk, thus $\mathsf{KeyGen}(msk, p)$ for PBS becomes $\mathsf{Delegate}(msk, p)$ for DPBS. To achieve stronger privacy notions, where the adversary is allowed to produce its own keys (see below), we define one more algorithm ChkKey, which on input pp, a signing key sk and a policy vector $\mathbf{p}$ outputs 1 iff sk is valid for the policies in $\mathbf{p}$.

We say that a scheme $\mathcal{DPBS}$ is *correct* relative to PC if for all $\lambda \in \mathbb{N}$, all $(pp, msk) \in [\mathsf{Setup}(1^\lambda)]$, all PC-valid $((p_1, \ldots, p_n), m, (w_1, \ldots, w_n))$, all $sk \in [\mathsf{Delegate}(\mathsf{Delegate}(\ldots \mathsf{Delegate}(msk, p_1) \ldots, p_{n-1}), p_n)]$ and all $\sigma \in [\mathsf{Sign}(sk, m, (w_1, \ldots, w_n))]$ we have $\mathsf{ChkKey}(pp, sk, \mathbf{p}) = 1$ and $\mathsf{Verify}(pp, m, \sigma) = 1$.

Unforgeability and indistinguishability. Unforgeability and indistinguishability are defined in the same spirit as for PBS. However, we strengthen indistinguishability by letting the adversary construct two keys under one of which the experiment makes a signature. (The adversary obtains msk in this experiment.) This models the case where Alice delegates different keys to Bob and Carol and should then not be able to tell by whom a message was signed. In order to prevent the adversary from outputting malfunctioning keys, the experiment first checks them using ChkKey. We refer to Appendix G.1 for the definitions.

Simulatability and extractability. Analogously to indistinguishability, we strengthen simulatability: the adversary outputs a signing key, which must pass ChkKey; it then receives either a signature under that key or a simulated signature and should not be able to tell the difference. (An oracle DELEGATE and an algorithm $\mathsf{SDelegate}$—analogous to KEY in $\mathbf{Exp}^{\text{SIM}}_{\mathcal{PBS}}$ and $\mathsf{SKeyGen}$ for PBS—would thus be redundant.) With $\mathbf{Exp}^{\text{SIM}}_{\mathcal{DPBS}}$ defined in Figure 6, we require that for every PT adversary $\mathcal{A}$: $\mathbf{Adv}^{\text{SIM}}_{\mathcal{DPBS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\text{SIM}}_{\mathcal{DPBS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}] - \frac{1}{2}$ is negligible in λ.

We also increase the power of the adversary $\mathcal{A}$ in the extractability game by allowing it to receive not only keys for policies of its choice, but also keys derived from others that are not revealed to $\mathcal{A}$ (and therefore do not restrict $\mathcal{A}$ in winning the game). $\mathcal{DPBS}$ is extractable if $\Pr[\mathbf{Exp}^{\text{EXT}}_{\mathcal{DPBS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}]$ is negligible in λ for every PT $\mathcal{A}$.

Construction. In the PBS schemes in Figures 3 and 4, a signing key sk_p is simply a signature from the authority on the associated policy p. We add delegation to PBS by replacing the signature with an *append-only signature* [KMPR05]. These signatures allow anyone holding a signature on a message p to create a signature on $p\|p'$ for any p'. One can thus append a new part to a signed message, but this is the only transformation allowed. Append-only signatures (AOS) can be constructed from any signature scheme. (See Appendix G.2 for a formal definition and a construction.) Holding a key, which is a signature on a vector of policies $\mathbf{p}$, a user can delegate the key after (possibly) appending a new policy. For simplicity, we give a construction of DPBS using SE-NIZKs, which is analogous to the scheme in Figure 4. We stress that we could also construct DPBS from ordinary signatures, public-key encryption and NIZKs by replacing the signature scheme in Figure 3 by the construction of AOS in Appendix G.2. Let $\mathcal{AOS}ig = (\mathsf{KeyGen}_{\text{sig}}, \mathsf{Append}_{\text{sig}}, \mathsf{Verify}_{\text{sig}})$ be an AOS scheme and for a policy checker PC let $\mathcal{NIZK} = (\mathsf{Setup}_{\text{nizk}}, \mathsf{Prove}, \mathsf{Verify}_{\text{nizk}}, \mathsf{SimSetup}_{\text{nizk}}, \mathsf{SimProve}, \mathsf{Extr}_{\text{nizk}})$ be a SE-NIZK for the following **NP**-relation:

$$((vk, m), (\mathbf{p}, c, \mathbf{w})) \in R_{\text{NP}} \iff \mathsf{Verify}_{\text{sig}}(vk, \mathbf{p}, c) = 1 \ \wedge \ ((\mathbf{p}, m), \mathbf{w}) \in \mathsf{PC}$$

We define $\mathcal{DPBS}[\mathcal{AOS}ig, \mathcal{NIZK}]$ via the following algorithms and prove Theorem 4 below in Appendix G.3.

$\underline{\mathsf{Setup}(1^\lambda)}$
$crs \leftarrow\!\!\$\ \mathsf{Setup}_{\text{nizk}}(1^\lambda)$; $(mvk, msk') \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$
Return $pp \leftarrow (crs, mvk)$, $msk \leftarrow (pp, \varepsilon, msk')$

$\underline{\mathsf{Delegate}((pp, \mathbf{p}, c), p')}$
$c' \leftarrow\!\!\$\ \mathsf{Append}_{\text{sig}}(mvk, c, p')$
Return $sk \leftarrow (pp, \mathbf{p}\|p', c')$

$\underline{\mathsf{ChkKey}((crs, mvk), (pp, \mathbf{q}, c), \mathbf{p})}$
If $pp \neq (crs, mvk)$ or $\mathbf{q} \neq \mathbf{p}$ then return 0
Return $\mathsf{Verify}_{\text{sig}}(mvk, \mathbf{q}, c)$

$\underline{\mathsf{Sign}(sk = ((crs, mvk), \mathbf{p}, c), m, \mathbf{w})}$
Return $\sigma \leftarrow\!\!\$\ \mathsf{Prove}(crs, (mvk, m), (\mathbf{p}, c, \mathbf{w}))$

$\underline{\mathsf{SimSetup}(1^\lambda)}$
$(crs, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}_{\text{nizk}}(1^\lambda)$
$(mvk, msk') \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$
Return $pp \leftarrow (crs, mvk)$, $msk \leftarrow (pp, \varepsilon, msk')$,
$tr_{\text{pbs}} \leftarrow (pp, tr)$

$\underline{\mathsf{SimSign}(((crs, mvk), tr), m)}$
Return $\sigma \leftarrow\!\!\$\ \mathsf{SimProve}(crs, tr, (mvk, m))$

$\underline{\mathsf{Extr}(((crs, mvk), tr), m, \sigma)}$
$(\mathbf{p}, c, \mathbf{w}) \leftarrow \mathsf{Extr}_{\text{nizk}}(tr, (mvk, m), \sigma)$; return $(\mathbf{p}, \mathbf{w})$

$\underline{\mathsf{Verify}(pp = (crs, mvk), m, \sigma)}$
Return $\mathsf{Verify}_{\text{nizk}}(crs, (mvk, m), \sigma)$

Theorem 4. *If $\mathcal{AOS}ig$ is unforgeable under chosen-message attack (as defined in Appendix G.2) and $\mathcal{NIZK}$ is a simulation-extractable NIZK (Appendix F.1) for $\mathcal{L}(R_{NP})$ then $\mathcal{DPBS}[\mathcal{AOS}ig, \mathcal{NIZK}]$ is a delegatable PBS for PC which satisfies simulatability and extractability.*

References

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236. Springer, August 2010.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013*, volume 8043 of *LNCS*, pages 500–518. Springer, August 2013.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, March 2008.

[BF13] Manuel Barbosa and Pooya Farshim. On the semantic security of functional encryption schemes. In *PKC 2013*, volume 7778 of *LNCS*, pages 143–161, 2013.

[BG89] Mihir Bellare and Shafi Goldwasser. New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 194–211. Springer, August 1989.

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. Cryptology ePrint Archive, Report 2013/401, 2013.

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 60–79. Springer, March 2006.

[BM05] Walid Bagga and Refik Molva. Policy-based cryptography and applications. In *Financial Cryptography and Data Security*, pages 72–87. Springer, 2005.

[BMS13] Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable functional signatures. Cryptology ePrint Archive, Report 2013/408, 2013.

[BMT13] Mihir Bellare, Sarah Meiklejohn, and Susan Thomson. Key-versatile signatures and applications: RKA, KDM and Joint Enc/Sig. Cryptology ePrint Archive, Report 2013/326, 2013.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, May 2003.

[BO13] Mihir Bellare and Adam O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In *Cryptology and Network Security - 12th International Conference, CANS 2013, Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2013.

[Boy07] Xavier Boyen. Mesh signatures. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 210–227. Springer, May 2007.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, May / June 2006.

[BS04] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 04*, pages 168–177. ACM Press, October 2004.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, March 2011.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. Cryptology ePrint Archive, Report 2013/352, 2013.

[CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. Cryptology ePrint Archive, Report 2013/179, 2013.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, May 2001.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, August 2006.

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, April 1991.

[DCIJ+13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013*, volume 8043 of *LNCS*, pages 519–535. Springer, August 2013.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DHLW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 613–631. Springer, December 2010.

[DMP87] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 52–72. Springer, August 1987.

[FP08] Georg Fuchsbauer and David Pointcheval. Anonymous proxy signatures. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 201–217. Springer, September 2008.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990.

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

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, December 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, April 2008.

[KMPR05] Eike Kiltz, Anton Mityagin, Saurabh Panjwani, and Barath Raghavan. Append-only signatures. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 434–445. Springer, July 2005.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. Cryptology ePrint Archive, Report 2013/379, 2013.

[MM13] Christian Matt and Ueli Maurer. A constructive approach to functional encryption. Cryptology ePrint Archive, Report 2013/559, 2013.

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

[MUO96] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy signatures for delegating signing operation. In *ACM CCS 96*, pages 48–57. ACM Press, March 1996.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, December 2001.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

A Proofs that SIM and EXT Imply IND and UF

Simulatability implies indistinguishability. Assuming an adversary $\mathcal{A}$ against indistinguishability, we construct an adversary $\mathcal{B}$ against the simulatability. $\mathcal{B}$ receives (pp, msk), sets $j \leftarrow 1$, chooses $d \leftarrow\!\!\$ \{0,1\}$ and runs $\mathcal{A}$ on (pp, msk). Whenever $\mathcal{A}$ queries $\mathsf{LR}(p_0, p_1, m, w_0, w_1)$, if $\mathsf{PC}((p_0, m), w_0) = 0$ or $\mathsf{PC}((p_1, m), w_1) = 0$, it returns $\perp$; otherwise it queries $sk_0 \leftarrow\!\!\$ \textsc{Key}(p_0)$ and $sk_1 \leftarrow\!\!\$ \textsc{Key}(p_1)$ and $\sigma_d \leftarrow\!\!\$ \textsc{Signature}(j + d, m, w_d)$ and sets $j \leftarrow j + 2$; it returns (σ_d, sk_0, sk_1) to $\mathcal{A}$. When $\mathcal{A}$ terminates outputting b', $\mathcal{B}$ outputs 1 if $(b' = d)$ and 0 otherwise.

proc INITIALIZE // $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$j \leftarrow 0\ ;\ Q_S \leftarrow \emptyset$
Return pp

proc MAKESK(p) // $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$
$j \leftarrow j+1\ ;\ Q[j][1] \leftarrow p$
$Q[j][2] \leftarrow\!\!\$\ \mathsf{KeyGen}(pp, msk, p)\ ;\ Q[j][3] \leftarrow \emptyset$

proc REVEALSK(i) // $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$
If $i \notin [j]$ then return $\bot$
$sk \leftarrow Q[i][2]\ ;\ Q[i][2] \leftarrow \bot\ ;$ return sk

proc SIGN(i, m, w) // $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$
If $i \notin [j]$ then return $\bot$
If $Q[i][2] = \bot$ then return $\bot$
$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(pp, Q[i][2], m, w)$
$Q_S \leftarrow Q_S \cup \{(m, \sigma)\}$; return σ

proc FINALIZE(m, σ) // $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$
If $\mathsf{Verify}(pp, m, \sigma) = 0$ then return false
$(p, w) \leftarrow \mathsf{Extr}(tr, m, \sigma)$
If $(m, \sigma) \in Q_S$ then return false
For $i = 1, \ldots, j$ do
  If $Q[i][1] = p$ and $Q[i][2] = \bot$
    If $\mathsf{PC}((p, m), w) = 1$ then return false
Return true

proc INITIALIZE // $\mathbf{Exp}^{\text{UF-(2)}}_{\mathcal{PBS}}$
$(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
$j \leftarrow 0\ ;\ Q_S \leftarrow \emptyset$
Return pp

proc MAKESK(p) // $\mathbf{Exp}^{\text{UF-(2)}}_{\mathcal{PBS}}$
$j \leftarrow j+1\ ;\ Q[j][1] \leftarrow p$
$Q[j][2] \leftarrow\!\!\$\ \mathsf{SKeyGen}(pp, tr, p)\ ;\ Q[j][3] \leftarrow \emptyset$

proc REVEALSK(i) // $\mathbf{Exp}^{\text{UF-(2)}}_{\mathcal{PBS}}$
If $i \notin [j]$ then return $\bot$
$sk \leftarrow Q[i][2]\ ;\ Q[i][2] \leftarrow \bot\ ;$ return sk

proc SIGN(i, m, w) // $\mathbf{Exp}^{\text{UF-(2)}}_{\mathcal{PBS}}$
If $i \notin [j]$ then return $\bot$
If $Q[i][2] = \bot$ then return $\bot$
If $\mathsf{PC}((Q[i][1], m), w) = 0$ then return $\bot$
$\sigma \leftarrow\!\!\$\ \mathsf{SimSign}(pp, tr, m)$
$Q_S \leftarrow Q_S \cup \{(m, \sigma)\}$; return σ

proc FINALIZE(m, σ) // $\mathbf{Exp}^{\text{UF-(2)}}_{\mathcal{PBS}}$
If $\mathsf{Verify}(pp, m, \sigma) = 0$ then return false
$(p, w) \leftarrow \mathsf{Extr}(tr, m, \sigma)$
If $(m, \sigma) \in Q_S$ then return false
For $i = 1, \ldots, j$ do
  If $Q[i][1] = p$ and $Q[i][2] = \bot$
    If $\mathsf{PC}((p, m), w) = 1$ then return false
Return true

Figure 7: Games $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$ and $\mathbf{Exp}^{\text{UF-(2)}}_{\mathcal{PBS}}$ in the proof SIM/EXT $\Rightarrow$ UF

If in $\mathbf{Exp}^{\text{SIM}}_{\mathcal{PBS},\mathcal{B}}$, the challenger's bit is 1 then $\mathcal{B}$ perfectly simulates $\mathbf{Exp}^{\text{IND}}_{\mathcal{PBS}}$ for $\mathcal{A}$; if on the other hand the bit is 0 then the bit d chosen by $\mathcal{B}$ is perfectly hidden from $\mathcal{A}$, meaning $\mathcal{B}$ outputs 1 with probability $\frac{1}{2}$. Together, this yields $\mathbf{Adv}^{\text{SIM}}_{\mathcal{PBS},\mathcal{B}} = \frac{1}{2} \cdot \mathbf{Adv}^{\text{IND}}_{\mathcal{PBS},\mathcal{A}}$.

Simulatability and extractability imply unforgeability. The benefit of defining unforgeability with the help of an extractor is that the experiment is efficiently decidable, as there are no more conditions like $(p, m) \in \mathcal{L}(\mathsf{PC})$. In efficiently decidable experiments, we can replace real signatures and keys by simulated ones without changing the adversary's behavior, since we could build a distinguisher that breaks the SIM.

Recall $\mathbf{Exp}^{\text{IND}}_{\mathcal{PBS}}$, defined in Figure 1. Now consider the modification $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$ given in Figure 7, whose FINALIZE procedure is basically that of $\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS}}$: the signature must be valid and (m, σ) must not be a response of the SIGN oracle. We then apply the extractor of $\mathcal{PBS}$ to get (p, w) and the adversary wins if either no secret key for p was ever revealed or if $\mathsf{PC}((p, m), w) = 0$.

Claim. UF-(1) $\Rightarrow$ UF.

We show how to use an adversary $\mathcal{A}$ winning UF to construct $\mathcal{B}$ which wins UF-(1). What makes the proof not quite straightforward is the fact that $\mathcal{A}$ might win UF by outputting a pair (m, σ) which is a query and response of the SIGN oracle,[3] whereas this would make UF-(1) immediately return false, since $(m, \sigma) \in Q_S$.

We start with the simpler case, where $\mathcal{A}$ never outputs a query/response pair from the SIGN oracle. Our reduction $\mathcal{B}$ forwards all messages between its $\mathbf{Exp}^{\text{UF-(1)}}_{\mathcal{PBS}}$-challenger and $\mathcal{A}$. If $\mathcal{A}$ wins UF by outputting

[3] We note that winning UF this way can only happen if there exist p^*, m^* and w^*, with $\mathsf{PC}((p^*, m^*), w^*) = 0$, but for which $\mathsf{Sign}(pp, \mathsf{KeyGen}(pp, msk, p^*), m^*, w^*)$ outputs a valid signature, which we have not explicitly excluded. However, by this very attack, this cannot happen in any scheme satisfying UF; neither in any scheme satisfying EXT and SIM, since we prove that they imply UF.

(m^*, σ^*) then we have (1) $\mathsf{Verify}(pp, m^*, \sigma^*) = 1$ and (2) $(m^*, \sigma^*) \notin Q_S$ (by assumption). Let $(\bar{p}, \bar{w}) \leftarrow \mathsf{Extr}(tr, m^*, \sigma^*)$. If $\mathsf{PC}((\bar{p}, m^*), \bar{w}) = 0$ then $\mathcal{B}$ wins. Otherwise, we have $(\bar{p}, m^*) \in \mathcal{L}(\mathsf{PC})$. Thus, by the winning condition of UF, for all i with $Q[i][1] = \bar{p}$ we must have $Q[i][2] \neq \perp$. Thus, in this case too $\mathcal{B}$ wins.

We now deal with adversaries winning UF by outputting $(m^*, \sigma^*) \in Q_S$. $\mathcal{B}$ guesses the index of the first SIGN query involving m^*. (That is, $\mathcal{B}$ picks a random SIGN query and if the queried message has been queried before then $\mathcal{B}$ aborts.) Let the guessed query be for (i^*, m^*, w^*). Instead of querying its own SIGN oracle, $\mathcal{B}$ does the following: If $Q[i^*][2] = \perp$ then $\mathcal{B}$ looks up the signing key sk^* it forwarded when $\mathcal{A}$ queried REVEALSK, otherwise $\mathcal{B}$ queries its own REVEALSK oracle to get sk^*. $\mathcal{B}$ computes $\sigma^* \leftarrow\!\!\$\ \mathsf{Sign}(sk^*, m^*, w^*)$ and returns σ^*. Every time $\mathcal{A}$ makes a SIGN query involving m^*, $\mathcal{B}$ also proceeds as just described. It is clear that this perfectly simulates $\mathcal{A}$'s oracles in UF.

Assume now that $\mathcal{A}$ wins by outputting (m^*, σ^*) and that $\mathcal{B}$ guessed correctly. Since $\mathcal{A}$ wins, we have (1) $\mathsf{Verify}(pp, m^*, \sigma^*) = 1$. Since $\mathcal{B}$ guessed correctly, no entry in Q_S starts with m^*, thus (2) $(m^*, \sigma^*) \notin Q_S$. Let $(\bar{p}, \bar{w}) \leftarrow \mathsf{Extr}(tr, m^*, \sigma^*)$. If no key for $\bar{p}$ was ever revealed then $\mathcal{B}$ wins. There are two ways that a key for $\bar{p}$ could have been revealed in UF-(1): either $\mathcal{A}$ queried REVEALSK for it, or $\mathcal{A}$ made a SIGN query involving m^* (in which case $\mathcal{B}$ may have queried REVEALSK). In either case, since $\mathcal{A}$ won UF, by its winning condition, we must have $(\bar{p}, m^*) \notin \mathcal{L}(\mathsf{PC})$. A forteriori $\mathsf{PC}(\bar{p}, m^*), \bar{w}) = 0$; thus, also in the case that keys for $\bar{p}$ were revealed, $\mathcal{B}$ wins.

Claim. EXT $\wedge$ SIM $\Rightarrow$ UF-(1).

By SIM, UF-(1) is indistinguishable from $\mathbf{Exp}^{\text{UF-(2)}}_{\mathcal{PBS}}$, given in Figure 7, where we replaced Setup by $\mathsf{SimSetup}$, KeyGen by $\mathsf{SKeyGen}$ and Sign by $\mathsf{SimSign}$ conditioned on the fact that the inputs satisfy PC. Note that it is because both games are efficiently decidable, that we can construct a distinguisher which uses an adversary behaving differently in the two games to break SIM.

It is now easily seen that from any $\mathcal{A}$ winning UF-(2) we can construct $\mathcal{B}$ winning EXT. When $\mathcal{A}$ queries MAKESK(p), $\mathcal{B}$ simply stores $(p, \emptyset, \emptyset)$ in Q. When $\mathcal{A}$ queries REVEALSK(i), $\mathcal{B}$ queries SKEYGEN$(Q[i][1])$. And whenever $\mathcal{A}$ queries SIGN(i, m, w), if $\mathsf{PC}((Q[i][1], m), w) = 1$ then $\mathcal{B}$ queries SIMSIGN(m). $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins.

B Security Proofs for the Construction in Section 4.1

Simulatability. We show that two runs of $\mathbf{Exp}^{\text{SIM}}_{\mathcal{PBS},\mathcal{A}}$, one with b set to 1, and one with b set to 0 are indistinguishable. We proceed by games: We start with $\mathbf{Exp}^{\text{SIM}|b=1}_{\mathcal{PBS},\mathcal{A}}$. Our first change is that we replace $crs \leftarrow\!\!\$\ \mathsf{Setup}_{\text{nizk}}(1^\lambda)$ by $crs \leftarrow\!\!\$\ \mathsf{SimSetup}_{\text{nizk}}(1^\lambda)$ and in calls to SIGNATURE, we replace π by a simulated proof. This is indistinguishable by the zero-knowledge property of $\mathcal{NIZK}$. Note that in this game, the proof π can be computed knowing only $(pk, mvk, C_p, C_s, C_w, ovk, m)$, but not the content of C_p, C_s and C_w nor their randomness.

In the next game, for the SIGNATURE calls, we replace the ciphertexts C_p and C_w by encryptions of 0 and C_s by an encryption of a signature $s \leftarrow\!\!\$\ \mathsf{Sign}_{\text{sig}}(msk, 0\|ovk)$. This is indistinguishable by IND-CPA of $\mathcal{PKE}$. In the final game, we replace the simulated CRS by a real CRS. This final game is $\mathbf{Exp}^{\text{SIM}|b=0}_{\mathcal{PBS},\mathcal{A}}$, which concludes the proof.

Extractability. This notion is reduced to the security of both $\mathcal{S}ig$ and $\mathcal{O}t\mathcal{S}ig$. We distinguish two types of adversaries. Type 1 returns a forgery $(m, (ovk, C_p, C_s, C_w, \pi, \tau))$ such that there was a reply of a SIMSIGN query with the same ovk, Type-2 adversaries return forgeries with a fresh ovk.

We use Type-1 adversaries to break strong unforgeability of $\mathcal{O}t\mathcal{S}ig$. Given ovk^* from our challenger, we simulate $\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS},\mathcal{A}}$ except that for a randomly guessed valid SIMSIGN query, say for $\hat{m}$, we use ovk^*, compute $\hat{C}_p$, $\hat{C}_s$, $\hat{C}_w$ and $\hat{\pi}$ as per $\mathsf{SimSign}$ and complete the signature by querying our one-time oracle on $(\hat{m}, \hat{C}_p, \hat{C}_s, \hat{C}_w, \hat{\pi})$ to get $\hat{\tau}$. Assume that the adversary $\mathcal{A}$ is of Type 1, then with non-negligible probability his forgery is of the form $(m, \sigma = (ovk^*, C_p, C_s, C_w, \pi, \tau))$. Since, for $\mathcal{A}$ to have won, (m, σ) must be different from all query/response pairs for SIMSIGN, in particular, it must be dif-

ferent from $(\hat{m}, (ovk^*, \hat{C}_p, \hat{C}_s, \hat{C}_w, \hat{\pi}, \hat{\tau}))$, in which we embedded ovk^*. Therefore $((m, C_p, C_s, C_w, \pi), \tau) \neq ((\hat{m}, \hat{C}_p, \hat{C}_s, \hat{C}_w, \hat{\pi}), \hat{\tau})$ and τ is a one-time forgery on (m, C_p, C_s, C_w, π).

We now use Type-2 adversaries to break unforgeability of $\mathcal{S}ig$. On receiving vk^* from our challenger, we run $crs \leftarrow\!\!\$\ \mathsf{Setup}_{\text{nizk}}(1^\lambda)$ and $(pk, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pke}}(1^\lambda)$. We run the adversary $\mathcal{A}$ on $pp := (crs, pk, vk^*)$. When $\mathcal{A}$ makes an SKeyGen query, we compute sk_p by querying our signing oracle on $1\|p$; when $\mathcal{A}$ makes a SimSign query, we use our oracle to get a signature s on $0\|ovk$ and use that as a witness for the proof π in our PBS σ.

Suppose $\mathcal{A}$ wins the game by outputting $(m, \sigma = (ovk, C_p, C_s, \pi, \tau))$, which satisfies Verify. Using Extr, we get $p \leftarrow \mathsf{Dec}(dk, C_p)$ and $w \leftarrow \mathsf{Dec}(dk, C_w)$; additionally, we compute $s \leftarrow \mathsf{Dec}(dk, C_s)$. As σ is valid, soundness of $\mathcal{NIZK}$ and correctness of $\mathcal{PKE}$ imply that either (i) s is a valid signature on $1\|p$ and $\mathsf{PC}((p, m), w) = 1$; or (ii) s is a valid signature on $0\|ovk$. In case (ii), s is a valid forgery on $0\|ovk$, since Type-2 adversaries use a one-time key which was never issued in a SimSign query. In case (i), since $\mathcal{A}$ wins and $\mathsf{PC}((p, m), w) = 1$, we must have $p \notin Q_K$, thus s is a valid forgery on $1\|p$.

C Details of the PBS Construction Based on GS Proofs

To construct $\mathcal{PBS}$, we let $(\mathsf{Setup}_{\text{gs}}, \mathsf{Prove}_{\text{gs}}, \mathsf{Verify}_{\text{gs}}, \mathsf{Extr}_{\text{gs}})$ denote the Groth-Sahai proof system for PPEs, and let $(\mathsf{KeyGen}_{\text{sp}}, \mathsf{Sign}_{\text{sp}}, \mathsf{Verify}_{\text{sp}})$ be a structure-preserving signature scheme that can sign vectors of group elements (e.g. one from [AFG+10]), for which we let $\mathsf{V}(vk, m, \sigma)$ denote the set of equations representing $\mathsf{Verify}_{\text{sp}}(vk, m, \sigma)$. Moreover, we let $(\mathsf{KeyGen}_{\text{ots}}, \mathsf{Sign}_{\text{ots}}, \mathsf{Verify}_{\text{ots}})$ be a strongly unforgeable one-time signature scheme whose verification keys are group elements (e.g. one from [AFG+10]). In the following we define a policy-based signature scheme which follows closely the generic construction and whose security is proved analogously. Since Groth-Sahai proofs are proofs of knowledge, we do not need to encrypt the witnesses as in the generic construction.

One technically is that we need to express the disjunction in R_{NP} from Equation (1) (Section 4.1) as the conjunction of sets of PPEs. We do so by following Groth's technique in [Gro06]. For an equation $\mathsf{E}(\underline{\mathbf{X}}, \underline{\mathbf{Y}})$ as defined in Equation (2) (Section 4.2), we define the set $\mathsf{E}^{(\text{sim})}(\underline{\mathbf{X}}, \underline{\mathbf{Y}})(\underline{Q'_1}, \ldots, \underline{Q'_k}, \underline{T})$ having additional variables $Q'_1, \ldots Q'_k, T$ as

$$\prod e(P_i, \underline{Q'_i}) \prod e(A_j, \underline{Y_j}) \prod e(\underline{X_i}, B_i) \prod \prod_{j=1}^{n} e(\underline{X_i}, \underline{Y_j})^{\gamma_{ij}} = 1 \quad \wedge \quad \bigwedge_i \ e(\underline{T}, \underline{Q'_i} \cdot Q_i^{-1}) = 1 \tag{5}$$

When $T = 1_{\mathbb{G}}$ then setting $X_i \leftarrow 1_{\mathbb{G}}$ and $Q'_i, Y_j \leftarrow 1_{\mathbb{H}}$, for all i, j satisfies $\mathsf{E}^{(\text{sim})}$. However, if $T \neq 1_{\mathbb{G}}$ then the only satisfying assignment to Q'_i is Q_i and thus $\mathbf{X}, \mathbf{Y}$ must satisfy the original equation E. In order to express a disjunction of two equations $\mathsf{E}_1(\underline{\mathbf{V}}, \underline{\mathbf{W}})$ and $\mathsf{E}_2(\underline{\mathbf{X}}, \underline{\mathbf{Y}})$, we convert them to $\mathsf{E}_1^{(\text{sim})}(\underline{\mathbf{V}}, \underline{\mathbf{W}})(\underline{\mathbf{Q}'}, \underline{T_1})$ and $\mathsf{E}_2^{(\text{sim})}(\underline{\mathbf{X}}, \underline{\mathbf{Y}})(\underline{\mathbf{Q}''}, \underline{T_2})$, respectively, and add the following equation E_T: $e(\underline{T_1} \cdot \underline{T_2} \cdot G^{-1}, H) = 1$. Now if we have a witness $(\mathbf{V}, \mathbf{W})$ for E_1, we can set $T_1 \leftarrow G$ and $T_2 \leftarrow 1_{\mathbb{G}}$ (and $\mathbf{X} \leftarrow \mathbf{1}_{\mathbb{G}}$ and $\mathbf{Q}'', \mathbf{Y} \leftarrow \mathbf{1}_{\mathbb{H}}$); whereas if we have a witness for E_2, we set $T_1 \leftarrow 1_{\mathbb{G}}$ (and all other variables in E_1 to 1) and $T_2 \leftarrow G$. In both cases we get a witness for the set $(\mathsf{E}_T, \mathsf{E}_1^{(\text{sim})}, \mathsf{E}_2^{(\text{sim})})$. However, E_T prevents us from setting both $\mathbb{T}_1 \leftarrow 1_{\mathbb{G}}$ and $\mathbb{T}_2 \leftarrow 1_{\mathbb{G}}$, which means we must have a witness for either E_1 or E_2. Note that this only works if the equations do not share any variables.

Policy checkers for PPEs. Consider a policy p describing (for simplicity) a single equation defined by $p = (\mathbf{P}, \mathbf{Q}, \mathbf{A}, \mathbf{B}, \mathbf{K}, \mathbf{L}, \Gamma = (\gamma_{ij}), \Delta = (\delta_{ij}), \Phi = (\phi_{ij}), \Psi = (\psi_{ij}))$. Let $m = (\mathbf{M}, \mathbf{N}) \in \mathbb{G}^{n_m} \times \mathbb{H}^{\ell_m}$ be a message and $w = (\mathbf{V}, \mathbf{W}) \in \mathbb{G}^{n_w} \times \mathbb{H}^{\ell_w}$ be a witness. Then the policy checker PC for our construction is defined as follows:

$$\begin{aligned} \mathsf{PC}((p, m), w) = 1 \iff & \prod e(P_i, Q_i) \prod e(A_j, N_j) \prod e(M_i, B_i) \prod e(K_j, W_j) \prod e(V_i, L_i) \\ & \prod \prod e(M_i, N_j)^{\gamma_{ij}} \prod \prod e(M_i, W_j)^{\delta_{ij}} \prod \prod e(V_i, N_j)^{\phi_{ij}} \prod \prod e(V_i, W_j)^{\psi_{ij}} = 1_{\mathbb{T}} \ , \end{aligned} \tag{6}$$

which is the most general form of a PPE over variables $\mathbf{M}, \mathbf{N}, \mathbf{V}$ and $\mathbf{W}$. Note that we assume that all policies are of a fixed length, since we cannot hide the *form* of the set of equations they define.

We start with expressing a policy in terms of group elements only. Abusing notation slightly, we replace the matrices $\Gamma = (\gamma_{i,j}), \Delta = (\delta_{ij}), \Phi = (\phi_{ij}), \Psi = (\psi_{ij}) \in \mathbb{Z}_p^{n\times\ell}$ in the policy description by matrices of $\mathbb{G}$-elements $\mathbf{\Gamma} = (\Gamma_{i,j} := G^{\gamma_{i,j}}), \mathbf{\Delta} = (\Delta_{i,j} := G^{\delta_{ij}}), \mathbf{\Phi} = (\Phi_{i,j} := G^{\phi_{ij}}), \mathbf{\Psi} = (\Psi_{i,j} := G^{\psi_{ij}}) \in \mathbb{G}^{n\times\ell}$. Henceforth, we assume that policies are given in this way.

Applying the transformation $\mathsf{E} \to \mathsf{E}^{(\text{no-exp})}$ from Equation (3) (Section 4.2) to (6), we obtain:

$$\begin{aligned}\mathsf{PC}^{(\text{no-exp})}((\underline{p},m),\underline{w}) = 1 \iff & \prod e(\underline{P_i},\underline{Q_i}) \prod e(\underline{A_j},N_j) \prod e(M_i,\underline{B_i}) \prod e(\underline{K_j},\underline{W_j}) \prod e(\underline{V_i},\underline{L_i}) \\ & \prod\prod e(M_i,\underline{\widehat{N}_{ij}^{(1)}}) \prod\prod e(M_i,\underline{\widehat{W}_{ij}^{(1)}}) \prod\prod e(\underline{V_i},\underline{\widehat{N}_{ij}^{(2)}}) \prod\prod e(\underline{V_i},\underline{\widehat{W}_{ij}^{(2)}}) = 1 \\ & \wedge \bigwedge_{i,j} e(G,\underline{\widehat{N}_{ij}^{(1)}}) = e(\underline{G^{\gamma_{ij}}},N_j) \wedge \bigwedge_{i,j} e(G,\underline{\widehat{W}_{ij}^{(1)}}) = e(\underline{G^{\delta_{ij}}},\underline{W_j}) \\ & \wedge \bigwedge_{i,j} e(G,\underline{\widehat{N}_{ij}^{(2)}}) = e(\underline{G^{\phi_{ij}}},N_j) \wedge \bigwedge_{i,j} e(G,\underline{\widehat{W}_{ij}^{(2)}}) = e(\underline{G^{\psi_{ij}}},\underline{W_j})\end{aligned}$$

where we introduced new variables $\widehat{N}_{ij}^{(1)}, \widehat{N}_{ij}^{(2)}, \widehat{W}_{ij}^{(1)}, \widehat{W}_{ij}^{(2)}$, and corresponding equations. Note that these equations contain the elements from $\mathbf{\Gamma}, \mathbf{\Delta}, \mathbf{\Psi}, \mathbf{\Phi}$ as variables.

It remains to make the above set simulatable, so they can be a clause of a disjunction. Since the above does not contain any pairing of constants, simulatability can be achieved easily by replacing equations of type $e(G,\underline{\widehat{Y}_{ij}}) = e(G^{\gamma_{ij}},\underline{Y_j})$ by $e(\underline{T},\underline{\widehat{Y}_{ij}}) = e(G^{\gamma_{ij}},\underline{Y_j})$. We get thus:

$$\begin{aligned}\mathsf{PC}^{(\text{sim})}((\underline{p},m),\underline{w})(\underline{T}) = 1 \iff & \prod e(\underline{P_i},\underline{Q_i}) \prod e(\underline{A_j},N_j) \prod e(M_i,\underline{B_i}) \prod e(\underline{K_j},\underline{W_j}) \prod e(\underline{V_i},\underline{L_i}) \\ & \prod\prod e(M_i,\underline{\widehat{N}_{ij}^{(1)}}) \prod\prod e(M_i,\underline{\widehat{W}_{ij}^{(1)}}) \prod\prod e(\underline{V_i},\underline{\widehat{N}_{ij}^{(2)}}) \prod\prod e(\underline{V_i},\underline{\widehat{W}_{ij}^{(2)}}) = 1 \\ & \wedge \bigwedge_{i,j} e(\underline{T},\underline{\widehat{N}_{ij}^{(1)}}) = e(\underline{\Gamma_{ij}},N_j) \wedge \bigwedge_{i,j} e(\underline{T},\underline{\widehat{W}_{ij}^{(1)}}) = e(\underline{\Delta_{ij}},\underline{W_j}) \\ & \wedge \bigwedge_{i,j} e(\underline{T},\underline{\widehat{N}_{ij}^{(2)}}) = e(\underline{\Phi_{ij}},N_j) \wedge \bigwedge_{i,j} e(\underline{T},\underline{\widehat{W}_{ij}^{(2)}}) = e(\underline{\Psi_{ij}},\underline{W_j}) \quad (7)\end{aligned}$$

We can now express the generic equation in (1) as a set of pairing-product equations $\mathsf{E}^{(\text{disj})}$ as follows. Since the clauses of the disjunction must not have common variables, we use S_1 and S_2 for the issuer's signatures; and in order to separate their domains, we prepend G^i for $i = 0, 1$ to their messages. We define

$$\begin{aligned}\mathsf{E}^{(\text{disj})}(mvk,\underline{p},\underline{S_1},\underline{S_2},\underline{w},ovk,m,\underline{T_1},\underline{T_2}) : & \\ & e(\underline{T_1}\cdot\underline{T_2}\cdot G^{-1},H) = 1 \quad \wedge \\ & \mathsf{V}^{(\text{sim})}(mvk,(G,\underline{\mathbf{P}},\underline{\mathbf{Q}},\underline{\mathbf{A}},\underline{\mathbf{B}},\underline{\mathbf{K}},\underline{\mathbf{L}},\underline{\mathbf{\Gamma}},\underline{\mathbf{\Delta}},\underline{\mathbf{\Phi}},\underline{\mathbf{\Psi}}),\underline{S_1})(\underline{T_1}) = 1 \quad \wedge \\ & \mathsf{PC}^{(\text{sim})}\big((\underline{\mathbf{P}},\underline{\mathbf{Q}},\underline{\mathbf{A}},\underline{\mathbf{B}},\underline{\mathbf{K}},\underline{\mathbf{L}},\underline{\mathbf{\Gamma}},\underline{\mathbf{\Delta}},\underline{\mathbf{\Phi}},\underline{\mathbf{\Psi}}),(\mathbf{M},\mathbf{N}),(\underline{\mathbf{V}},\underline{\mathbf{W}})\big)(\underline{T_1}) = 1 \quad \wedge \\ & \mathsf{V}^{(\text{sim})}(mvk,(1,ovk),\underline{S_2})(\underline{T_2}) = 1\end{aligned}$$

When making a PBS, a user has to construct a proof for $\mathsf{E}^{(\text{disj})}$. Since she does not have a valid signature S_2, she must set $T_2 \leftarrow 1_{\mathbb{G}}$ in order to simulate the last equation in $\mathsf{E}^{(\text{disj})}$. To satisfy the 1st equation, she must set $T_1 \leftarrow G$, which means she must have a signature S_1 on a policy p in order to satisfy the 2nd equation. This however fixes the values $\mathbf{P}, \mathbf{Q}, \mathbf{A}, \mathbf{B}, \mathbf{K}, \mathbf{L}, \mathbf{\Gamma}, \mathbf{\Delta}, \mathbf{\Psi}, \mathbf{\Phi}$ in the 3rd equation $\mathsf{PC}^{(\text{sim})}$ to her policy. In order to satisfy $\mathsf{PC}^{(\text{sim})}$ in (7), she now needs to know $\mathbf{M}, \mathbf{N}, \mathbf{V}, \mathbf{W}$ satisfying the original equations and sets $\widehat{N}_{ij}^{(1)} \leftarrow N_j^{\gamma_{ij}}$, $\widehat{N}_{ij}^{(2)} \leftarrow N_j^{\phi_{ij}}$, $\widehat{W}_{ij}^{(1)} \leftarrow W_j^{\delta_{ij}}$, $\widehat{W}_{ij}^{(2)} \leftarrow W_j^{\psi_{ij}}$ (since $T = G$).

The simulator, on the other hand, produces a signature S_2 satisfying the last equation in $\mathsf{E}^{(\text{disj})}$, and can then set $T_2 \leftarrow G$ and $T_1 \leftarrow 1_{\mathbb{G}}$. The latter enables the simulator to set all other variables to 1, which yields a satisfying assignment for $\mathsf{E}^{(\text{disj})}$.

Now that we have defined all the required concepts, our construction of PBS using Groth-Sahai proofs and structure-preserving signatures is quite straightforward. We present it in Figure 8. Security is proven analogously to that of the scheme in Figure 3. Extractability follows from perfect extractability of GS proofs, unforgeability of $\mathcal{S}ig_{\text{sp}}$ and strong unforgeability of $\mathcal{O}t\mathcal{S}ig$ (note that we have encoded policies as group elements, which can be extracted from GS proofs, whereas exponents cannot). Simulatability follows

$\underline{\mathsf{Setup}(1^\lambda)}$
$(crs, xk) \leftarrow\!\!\$\ \mathsf{Setup}_{\mathrm{gs}}(1^\lambda)$
$(mvk, msk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{sp}}(1^\lambda)$
Return $pp \leftarrow (crs, mvk)$ and msk

$\underline{\mathsf{KeyGen}(msk, p)}$
$S \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{sp}}(msk, (G, p))$
Return $sk_p \leftarrow (pp, p, S)$

$\underline{\mathsf{Sign}(sk_p, m, w)}$
Parse $((crs, mvk), p, S) \leftarrow sk_p$
If $\mathsf{PC}((p, m), w) = 0$ then return $\bot$
$(ovk, osk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{ots}}(1^\lambda)$
$\pi \leftarrow\!\!\$\ \mathsf{Prove}_{\mathrm{gs}}(crs, \mathsf{E}^{(\mathrm{disj})}(mvk, \underline{p}, \underline{S}, \underline{\mathbf{1}}, \underline{w}, ovk, m, \underline{G}, \underline{1_{\mathbb{G}}}))$
$\tau \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{ots}}(osk, (m, \pi))$
Return $\sigma \leftarrow (ovk, \pi, \tau)$

$\underline{\mathsf{Verify}(pp, m, \sigma)}$
Parse $(crs, mvk) \leftarrow pp$; $(ovk, \pi, \tau) \leftarrow \sigma$
Return 1 iff
$\mathsf{Verify}_{\mathrm{gs}}(crs, (crs, \mathsf{E}^{(\mathrm{disj})}(mvk, \underline{\cdot}, \underline{\cdot}, \underline{\cdot}, \underline{\cdot}, ovk, m, \underline{\cdot}, \underline{\cdot})), \pi) = 1$
and $\mathsf{Verify}_{\mathrm{ots}}(ovk, (m, \pi), \tau) = 1$

$\underline{\mathsf{SimSetup}(1^\lambda)}$
$(crs, xk) \leftarrow\!\!\$\ \mathsf{Setup}_{\mathrm{gs}}(1^\lambda)$
$(mvk, msk) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$
Return $pp \leftarrow (crs, mvk)$, msk, $tr \leftarrow (msk, xk)$

$\underline{\mathsf{SKeyGen}((msk, xk), p)}$
$S \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{sp}}(msk, (G, p))$
Return $sk_p \leftarrow (pp, p, s)$

$\underline{\mathsf{SimSign}((msk, xk), m)}$
$(ovk, osk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{ots}}(1^\lambda)$
$S \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{sp}}(msk, (1, ovk))$
$\pi \leftarrow\!\!\$\ \mathsf{Prove}_{\mathrm{gs}}(crs, \mathsf{E}^{(\mathrm{disj})}(mvk, \underline{\mathbf{1}}, \underline{\mathbf{1}}, \underline{S}, \underline{\mathbf{1}}, ovk, m, \underline{1_{\mathbb{G}}}, \underline{G}))$
$\tau \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{ots}}(osk, (m, \pi))$
Return $\sigma \leftarrow (ovk, \pi, \tau)$

$\underline{\mathsf{Extr}((msk, xk), m, \sigma)}$
Parse $(ovk, \pi, \tau) \leftarrow \sigma$
$(p, S_1, S_2, w, T_1, T_2) \leftarrow \mathsf{Extr}_{\mathrm{gs}}(xk, \pi)$
Return (p, w)

Figure 8: Construction of PBS with Groth-Sahai proofs and structure-preserving signatures

from witness indistinguishability of Groth-Sahai proofs: since we directly use a proof of knowledge, we need not simulate proofs as there are no ciphertexts, but instead simply change the witness used by Sign to the witness used by SimSign.

D Attribute-Based Signatures from Policy-Based Signatures

In attribute-based signatures (ABS) a trusted setup produces parameters and a master secret key. The latter is then used to issue keys for sets of attributes from a universe $\mathbb{A}$. Now a holder of such a key can sign messages w.r.t. to a predicate Υ over attributes, which must evaluate to 1 on the set of attributes for the key. The predicate is in the clear, so verification of a signature is w.r.t. the predicate.

The model. In view of a generalization to multiple authorities, Maji et al. [MPR11] separate the setup algorithm into a trusted setup TSetup, which outputs public parameters tpk and ASetup, run by the attribute-issuing authority, which outputs a public/private key pair (apk, ask).

For the single-authority case (which we consider[4]), TSetup and ASetup can be combined to Setup without weakening security: In the unforgeability game, both TSetup and ASetup are run by the experiment impersonating the attribute-issuing authority. Moreover, even though not explicitly stated, privacy also requires the pair $(apk, ask) \leftarrow\!\!\$\ \mathsf{ASetup}$ to be set up honestly.[5] For privacy, the adversary impersonates thus an "honest but curious" authority, which we model by giving the adversary the authority's secret key. Since security of our scheme is extraction-based, we cannot hope to achieve *perfect* privacy (meaning signatures produced with keys for different sets of attributes are distributed equally); we thus give a computational analog. We give a formal definition of the model.

We denote the message space by $\mathbb{M}$. A scheme $\mathcal{ABS}$ is parametrized by an *attribute universe* $\mathbb{A}$. A *claim predicate* over $\mathbb{A}$ is a monotone boolean function $\Upsilon\colon \mathcal{P}(\mathbb{A}) \to \{0, 1\}$. On input the security

[4] In ABS, different attributes could be issued by different authorities; in PBS however, a key only corresponds to one policy.

[5] If ask is maliciously set up so that AttrGen outputs a working key for one set of attributes and a key leading to invalid signatures for another set then privacy does not hold.

$\mathbf{Exp}^{\text{PRIV}}_{\mathcal{ABS}}$

proc INITIALIZE
$(pp, ask) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$; $b \leftarrow\!\!\$\ \{0,1\}$
Return (pp, ask)

proc LR(A_0, A_1, m, Υ)
If $\Upsilon(A_0) = 0$ or $\Upsilon(A_1) = 0$
then return $\bot$
$sk_0 \leftarrow\!\!\$\ \mathsf{AttrGen}(ask, A_0)$
$sk_1 \leftarrow\!\!\$\ \mathsf{AttrGen}(ask, A_1)$
$\sigma_b \leftarrow\!\!\$\ \mathsf{Sign}(sk_b, m, \Upsilon)$
Return (σ_b, sk_0, sk_1)

proc FINALIZE(b')
Return $(b = b')$

$\mathbf{Exp}^{\text{UF}}_{\mathcal{ABS}}$

proc INITIALIZE
$(pp, ask) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$Q_K \leftarrow \emptyset$; $Q_S \leftarrow \emptyset$; return pp

proc ATTRGEN(A)
$Q_K \leftarrow Q_K \cup \{A\}$; return $\mathsf{AttrGen}(ask, A)$

proc SIGN(m, Υ)
$Q_S \leftarrow Q_S \cup \{(m, \Upsilon)\}$
Choose arbitrary $A \subseteq \mathbb{A}$, with $\Upsilon(A) = 1$
$sk_A \leftarrow\!\!\$\ \mathsf{AttrGen}(ask, A)$; return $\mathsf{Sign}(sk_A, m, \Upsilon)$

proc FINALIZE(m, Υ, σ)
If $\mathsf{Verify}(pp, m, \Upsilon, \sigma) = 0$ then return false
If $(m, \Upsilon) \notin Q_S$ and $\forall\, A \in Q_K$: $\Upsilon(A) = 0$
then return true ; else return false.

Figure 9: Games defining privacy and unforgeability for attribute-based signatures

parameter 1^λ, Setup outputs public parameters pp and the authority's secret key ask. On input ask and $A \subseteq \mathbb{A}$, $\mathsf{AttrGen}$ outputs a secret key sk_A. On input sk_A, m, Υ, where $\Upsilon(A) = 1$, Sign outputs a signature σ. On input pp, m, Υ, σ, Verify outputs a bit. Correctness requires that for all $\lambda \in \mathbb{N}$, all $(pp, ask) \in [\mathsf{Setup}(1^\lambda)]$, all $m \in \mathbb{M}$, all $A \subseteq \mathbb{A}$, all Υ with $\Upsilon(A) = 1$ and all $\sigma \in [\mathsf{Sign}(\mathsf{AttrGen}(ask, A), m, \Upsilon)]$, we have $\mathsf{Verify}(pp, m, \Upsilon, \sigma) = 1$.

We say that $\mathcal{ABS}$ has *privacy* if $\mathbf{Adv}^{\text{PRIV}}_{\mathcal{ABS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\text{PRIV}}_{\mathcal{ABS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}] - \frac{1}{2}$ is negligible in λ for all PT $\mathcal{A}$, with $\mathbf{Exp}^{\text{PRIV}}_{\mathcal{ABS},\mathcal{A}}$ defined on the left of Figure 9. On the right we define the game $\mathbf{Exp}^{\text{UF}}_{\mathcal{ABS},\mathcal{A}}$ and say that $\mathcal{ABS}$ is *unforgeable* if $\mathbf{Adv}^{\text{UF}}_{\mathcal{ABS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\text{UF}}_{\mathcal{ABS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}]$ is negligible in λ for all PT $\mathcal{A}$. Although our notion of privacy is only computational, it is stronger in another aspect: the adversary gets the two signing keys.[6]

Construction. Let $\mathbb{Y}$ denote the set of all monotone boolean functions over $\mathbb{A}$. We define a policy checker PC for our policy-based signature $\mathcal{PBS}$ which instantiates ABS. A policy is a set $A \subseteq \mathbb{A}$ and a message for $\mathcal{PBS}$ is in $\mathbb{Y} \times \mathbb{M}$, i.e., a claim predicate and the actual message. A $\mathcal{PBS}$ message satisfies a policy if the set of attributes defining the policy satisfy the predicate contained in the message. The policy checker PC for $\mathcal{PBS}$ is efficiently decidable (thus no witnesses are required) and is defined as: $\mathsf{PC}\colon \mathbb{A} \times (\mathbb{Y} \times \mathbb{M}) \to \{0,1\}$ with $(A, (\Upsilon, m)) \mapsto \Upsilon(A)$. Let $\mathcal{PBS} = (\mathsf{Setup}_{\text{pbs}}, \mathsf{KeyGen}_{\text{pbs}}, \mathsf{Sign}_{\text{pbs}}, \mathsf{Verify}_{\text{pbs}})$ be a policy-based signature scheme for PC. We define $\mathcal{ABS}[\mathcal{PBS}] = (\mathsf{Setup}, \mathsf{AttrGen}, \mathsf{Sign}, \mathsf{Verify})$ as follows:

$\underline{\mathsf{Setup}(1^\lambda)}$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}_{\text{pbs}}(1^\lambda)$; return pp, $ask \leftarrow msk$

$\underline{\mathsf{AttrGen}(ask, A)}$
Return $sk_A \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pbs}}(ask, A)$

$\underline{\mathsf{Sign}(pp, sk_A, m, \Upsilon)}$
Return $\mathsf{Sign}_{\text{pbs}}(sk_A, (\Upsilon, m))$

$\underline{\mathsf{Verify}(pp, m, \Upsilon, \sigma)}$
Return $\mathsf{Verify}_{\text{pbs}}(pp, (\Upsilon, m), \sigma)$

Theorem 5. *If $\mathcal{PBS}$ is a policy-based signature scheme satisfying indistinguishability and unforgeability then $\mathcal{ABS}[\mathcal{PBS}]$ is an attribute-based signature scheme satisfying privacy and unforgeability.*

Proof. First we provide the proof of privacy. Let $\mathcal{A}$ be an adversary against privacy of $\mathcal{ABS}$. It is quite straightforward to build $\mathcal{B}$ breaking indistinguishability of $\mathcal{PBS}$. $\mathcal{B}$ receives (pp, msk) from its challenger and forwards them to $\mathcal{A}$ as (pp, ask). When $\mathcal{A}$ queries LR(A_0, A_1, m, Υ), $\mathcal{B}$ queries its own LR oracle for the two policies $p_0 := A_0$ and $p_1 := A_1$, and the message (Υ, m), and forwards the response (σ, sk_0, sk_1) to $\mathcal{A}$.

[6] For group signatures, this notion was termed *full* anonymity [BMW03], as opposed to *selfless* anonymity [BS04], where users are able to recognize signatures produced with their own signing key.

proc INITIALIZE // $\mathbf{Exp}^{\mathrm{TRC}}_{\mathcal{GS}}$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{pke}}(1^\lambda)$; $Q_C \leftarrow \emptyset$; $Q_S \leftarrow \emptyset$
For $i = 1, \ldots, n$ do
  $sk_i \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{pbs}}(msk, (ek, i))$
  $\boldsymbol{gsk}[i] \leftarrow (pp, ek, i, sk_i)$
Return $((pp, ek), dk)$

proc FINALIZE$(m, (c, \sigma))$ // $\mathbf{Exp}^{\mathrm{TRC}}_{\mathcal{GS}}$
If $\mathsf{Verify}(pp, (c, m), \sigma) = 0$ then return false
$i^* \leftarrow \mathsf{Dec}(dk, c)$
If $i^* \in [n]$, $i^* \notin Q_C$ and $(i^*, m) \notin Q_S$ then return true
Return false

proc CORRUPT(i) // $\mathbf{Exp}^{\mathrm{TRC}}_{\mathcal{GS}}$
$Q_C \leftarrow Q_C \cup \{i\}$
Return $\boldsymbol{gsk}[i]$

proc GSIG(i, m) // $\mathbf{Exp}^{\mathrm{TRC}}_{\mathcal{GS}}$
$Q_S \leftarrow Q_S \cup \{(i, m)\}$
$r \leftarrow\!\!\$\ \{0,1\}^\lambda$
$c \leftarrow \mathsf{Enc}(ek, i; r)$
$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk_i, (c, m), r)$
Return (c, σ)

Figure 10: Games $\mathbf{Exp}^{\mathrm{TRC}}_{\mathcal{GS}}$ for the proof of traceability of $\mathcal{GS}$

For $i \in \{0,1\}$, we have $\mathsf{PC}(p_i, (\Upsilon, m)) = 0$ iff $\Upsilon(A_i) = 0$, thus $\mathcal{B}$'s oracle returns $\perp$, whenever $\mathcal{A}$'s oracle should return $\perp$. Otherwise $\mathcal{B}$'s oracle computes $sk_i \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{pbs}}(msk, p_i)$, for $i \in \{0,1\}$ and $\sigma_b \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{pbs}}(sk_b, (\Upsilon, m))$, which, by definition is the same as $sk_i \leftarrow\!\!\$\ \mathsf{AttrGen}(ask, A_i)$, for $i \in \{0,1\}$, and $\sigma_b \leftarrow\!\!\$\ \mathsf{Sign}(sk_b, m, \Upsilon)$. Thus $\mathcal{A}$ receives what it expects from its oracle LR. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. It is clear that $\mathcal{B}$ guesses b correctly whenever $\mathcal{A}$ does; thus, $\mathbf{Adv}^{\mathrm{IND}}_{\mathcal{PBS},\mathcal{B}} = \mathbf{Adv}^{\mathrm{PRIV}}_{\mathcal{ABS},\mathcal{A}}$.

Next we provide the proof of unforgeability. Assume $\mathcal{A}$ wins $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{ABS}}$; we construct $\mathcal{B}$ which wins $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{PBS}}$ with the same probability. $\mathcal{B}$ receives pp, forwards it to $\mathcal{A}$ and answers $\mathcal{A}$'s oracle queries as follows, after initializing a counter $j \leftarrow 1$. ATTRGEN(A): $\mathcal{B}$ queries MAKESK(A) and REVEALSK(j), forwards the received key sk_A to $\mathcal{A}$ and sets $j \leftarrow j + 1$. SIGN(m, Υ): $\mathcal{B}$ picks a random $A \subseteq \mathbb{A}$ with $\Upsilon(A) = 1$, queries MAKESK(A) and SIGN$(j, (\Upsilon, m))$, forwards the received signature to $\mathcal{A}$ and sets $j \leftarrow j + 1$. When $\mathcal{A}$ outputs (m, Υ, σ), $\mathcal{B}$ outputs $((\Upsilon, m), \sigma)$. Suppose $\mathcal{A}$ wins by outputting (m, Υ, σ). Then $1 = \mathsf{Verify}(pp, m, \Upsilon, \sigma) = \mathsf{Verify}_{\mathrm{pbs}}(pp, (\Upsilon, m), \sigma)$. Since $\mathcal{A}$ won, we have (1) $(m, \Upsilon) \notin Q_S$ and (2) for all $A \in Q_K : \Upsilon(A) = 0$. Thus in $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{PBS}}$, we have that for all i, for which $\mathsf{PC}(Q[i][1], (\Upsilon, m)) = \Upsilon(A) = 1$: $Q[i][2] \neq \perp$ (as otherwise $Q[i][1]$ would be in Q_K), and $(\Upsilon, m) \notin Q[i][3]$ (as otherwise (m, Υ) would be in Q_S). Thus, whenever $\mathcal{A}$ wins $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{ABS}}$, $\mathcal{B}$ wins $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{PBS}}$. □

Using the strategy to express disjunctions of statements as sets of pairing-product equations outlined in Section 4.2, we can express PC as a set of PPEs and thus use our efficient PBS-implementation to construct an efficient ABS. We conclude by remarking that we could also instantiate *key-policy* ABS, where the key is associated with a predicate and the message with a set of attributes. We would simply define $\mathsf{PC}\colon \mathbb{Y} \times (\mathbb{A} \times \mathbb{M}) \to \{0,1\}$, $\mathsf{PC}(\Upsilon, (A, m)) = \Upsilon(A)$.

E Proofs for the Construction of Group Signatures from PBS

Traceability from UF. Plugging in the definition of $\mathcal{GS}[\mathcal{PKE}, \mathcal{PBS}]$ in $\mathbf{Exp}^{\mathrm{TRC}}_{\mathcal{GS},\mathcal{A}}$, we get (after some simplification) the experiment in Figure 10.

We build an adversary $\mathcal{B}$ against UF of $\mathcal{PBS}$ which simulates the above game and wins $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{PBS}}$ whenever $\mathcal{A}$ wins $\mathbf{Exp}^{\mathrm{TRC}}_{\mathcal{GS}}$.

$\mathcal{B}(pp :$ MAKESK$(\cdot)$, REVEALSK$(\cdot)$, SIGN$(\cdot,\cdot,\cdot))$
  $(ek, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{pke}}(1^\lambda)$
  For $i = 1, \ldots, n$ do
    MAKESK$((ek, i))$
  $(m, (c, \sigma)) \leftarrow\!\!\$\ \mathcal{A}((pp, ek), dk) : \mathrm{CORRUPT}_{\mathcal{B}}(\cdot), \mathrm{GSIG}_{\mathcal{B}}(\cdot,\cdot))$
  Return $(m, (c, \sigma))$

CORRUPT$_{\mathcal{B}}(i)$
  $sk_i \leftarrow$ REVEALSK(i); return (pp, ek, i, sk_i)

GSIG$_{\mathcal{B}}(i, m)$
  $r \leftarrow\!\!\$\ \{0,1\}^\lambda$; $c \leftarrow \mathsf{Enc}(ek, i; r)$
  $\sigma \leftarrow\!\!\$\ $ SIGN$(i, (c, m), r)$; return (c, σ)

proc INITIALIZE // $\mathbf{Exp}^{\text{anon-}b}_{\mathcal{GS}}$, $\boxed{\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}}$

$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$; $Q \leftarrow \emptyset$

$\boxed{(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)\ ;\ Q \leftarrow \emptyset}$

$(ek, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pke}}(1^\lambda)$

For $i = 1, \ldots, n$

  $sk_i \leftarrow\!\!\$\ \mathsf{KeyGen}(msk, (ek, i))$

  $\boxed{sk_i \leftarrow\!\!\$\ \mathsf{SKeyGen}(msk, (ek, i))}$

  $\boldsymbol{gsk}[i] \leftarrow (pp, ek, i, sk_i)$

Return $((pp, ek), \boldsymbol{gsk})$

proc LR(i_0, i_1, m) // $\mathbf{Exp}^{\text{anon-}b}_{\mathcal{GS}}$, $\boxed{\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}}$

$r \leftarrow\!\!\$\ \{0,1\}^\lambda$; $c^* \leftarrow \mathsf{Enc}(ek, i_b; r)$

$\sigma^* \leftarrow\!\!\$\ \mathsf{Sign}(sk_i, (c^*, m), r)$

$\boxed{\sigma^* \leftarrow\!\!\$\ \mathsf{SimSign}(tr, (c^*, m))}$

$Q \leftarrow Q \cup \{(c^*, \sigma^*)\}$; return (c^*, σ^*)

proc OPEN(m, γ) // $\mathbf{Exp}^{\text{anon-}b}_{\mathcal{GS}}$, $\boxed{\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}}$

If $\gamma \in Q$ then return $\perp$

If $\mathsf{Verify}(pp, (c, m), \sigma) = 0$ then return $\perp$

Return $\mathsf{Dec}(dk, c)$

proc FINALIZE(b') // $\mathbf{Exp}^{\text{anon-}b}_{\mathcal{GS}}$, $\boxed{\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}}$

Return $(b = b')$

proc INITIALIZE // $\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS}}$

$(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$; $Q \leftarrow \emptyset$

$(ek, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{pke}}(1^\lambda)$

For $i = 1, \ldots, n$

  $sk_i \leftarrow\!\!\$\ \mathsf{SKeyGen}(msk, (ek, i))$

  $\boldsymbol{gsk}[i] \leftarrow (pp, ek, i, sk_i)$

Return $((pp, ek), \boldsymbol{gsk})$

proc LR(i_0, i_1, m) // $\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS}}$

$r \leftarrow\!\!\$\ \{0,1\}^\lambda$; $c^* \leftarrow \mathsf{Enc}(ek, 0; r)$

$\sigma^* \leftarrow\!\!\$\ \mathsf{SimSign}(tr, (c^*, m))$

$Q \leftarrow Q \cup \{(c^*, \sigma^*)\}$; return (c^*, σ^*)

proc OPEN(m, γ) // $\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS}}$

If $\gamma \in Q$ then return $\perp$

If $\mathsf{Verify}(pp, (c, m), \sigma) = 0$ then return $\perp$

Return $\mathsf{Dec}(dk, c)$

proc FINALIZE(b') // $\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS}}$

Return $(b = b')$

Figure 11: Games $\mathbf{Exp}^{\text{anon-}b}_{\mathcal{GS}}$, $\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}$ and $\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS}}$ for the proof of anonymity of $\mathcal{GS}$

Suppose that $\mathcal{A}$ wins $\mathbf{Exp}^{\text{TRC}}_{\mathcal{GS}}$ by outputting $(m^*, (c^*, \sigma^*))$. Thus $\mathsf{Verify}(pp, (c^*, m^*), \sigma^*) = 1$ and with $i^* \leftarrow \mathsf{Dec}(sk, c^*)$, we have that $i^* \in [n]$ and $\mathcal{A}$ has queried neither CORRUPT(i^*) nor GSIG(i^*, m^*). This means that $\mathcal{B}$ has neither queried REVEALSK(i^*) nor SIGN$(i^*, (c, m^*), r)$ for any c and r, thus $(*)$ $Q[i^*][2] \neq \perp$ and for all c: $(c, m^*) \notin Q[i][3]$, thus in particular $(c^*, m^*) \notin Q[i][3]$.

$\mathcal{B}$ wins $\mathbf{Exp}^{\text{UF}}_{\mathcal{PBS}}$ if $\mathsf{Verify}(pp, (c^*, m^*), \sigma^*) = 1$ and if for all i, for which $(Q[i][1], (c^*, m^*)) \in \mathcal{L}(\mathsf{PC})$, we have (1) $Q[i][2] \neq \perp$ and (2) $(c^*, m^*) \notin Q[i][3]$. Now for any (c, m), we have: $(Q[i][1], (c, m)) \in \mathcal{L}(\mathsf{PC})$ iff c is in the range of $\mathsf{Enc}(ek, i; \cdot)$, which, by correctness of $\mathcal{PKE}$, implies $i = \mathsf{Dec}(dk, c)$. Thus we only have to show (1) and (2) for $i \leftarrow i^*$; which we have already done in $(*)$.

Anonymity. We let $\mathbf{Exp}^{\text{anon-}d}_{\mathcal{GS}}$ denote the experiment $\mathbf{Exp}^{\text{ANON}}_{\mathcal{GS}}$ when the bit b is fixed to $b \leftarrow d$. To show anonymity it then suffices to prove that for all PT adversaries $\mathcal{A}$ the difference

$$\left|\Pr[\mathbf{Exp}^{\text{anon-1}}_{\mathcal{GS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}] - \Pr[\mathbf{Exp}^{\text{anon-0}}_{\mathcal{GS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}]\right|$$

is negligible in λ, which we will show by a series of game hops. Plugging our scheme into the definition $\mathbf{Exp}^{\text{anon-}b}_{\mathcal{GS}}$ (see Figure 5), we get the game given on the left of Figure 11, when we ignore the boxes.

We modify this game to $\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}$ by including the lines in boxes, which replace each respective preceding line. We show that $\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}$ is indistinguishable from $\mathbf{Exp}^{\text{anon-}b}_{\mathcal{GS}}$: An adversary which behaved differently in the two games could be used to break the notion SIM of $\mathcal{PBS}$ by building an adversary $\mathcal{B}$ as follows: $\mathcal{B}$ gets (pp, msk) from its challenger and uses its oracle KEY to compute sk_i and oracle SIGNATURE to compute σ in the boxed lines (or the ones above, depending on which game $\mathcal{B}$ plays). (Note that when $\mathcal{B}$ makes a SIGNATURE$(i, (c, m), r)$ query when answering the LR query then by definition we have $\mathsf{PC}\big(((ek, i), (c, m)), r\big) = 1$, so SIGNATURE returns the output of either $\mathsf{SimSign}$ or Sign.)

We next define $\mathbf{Exp}^{\text{anon-(2)}}$, given on the right in Figure 11, in which c is an encryption of 0 instead of i_b. Note that this final game is independent of the bit b, thus the adversary's probability of outputting b is $\frac{1}{2}$.

$\underline{\mathcal{B}^{(b)}(ek : \mathrm{LR}(\cdot,\cdot), \textsc{Dec}(\cdot))}$
$(pp, msk, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)$
For $i = 1, \ldots, n$
$sk_i \leftarrow\!\!\$\ \mathsf{SKeyGen}(msk, (ek, i))$
$\boldsymbol{gsk}[i] \leftarrow (pp, ek, i, sk_i)$
$b' \leftarrow\!\!\$\ \mathcal{A}((pp, ek), \boldsymbol{gsk} : \mathrm{LR}_{\mathcal{B}^{(b)}}(\cdot,\cdot,\cdot), \textsc{Open}_{\mathcal{B}^{(b)}}(\cdot,\cdot))$
Return b'

$\underline{\mathrm{LR}_{\mathcal{B}^{(b)}}(i_0, i_1, m)}$
$c^* \leftarrow\!\!\$\ \mathrm{LR}(0, i_b)$
$\sigma^* \leftarrow\!\!\$\ \mathsf{SimSign}(tr, (c^*, m))$
Return (c^*, σ^*)

$\underline{\textsc{Open}_{\mathcal{B}^{(b)}}(m, (c, \sigma))}$
If $\mathsf{Verify}(pp, (c, m), \sigma) = 0$ then return $\bot$
If $c = c^*$ and $\sigma \neq \sigma^*$ (QRY)
$\mathcal{B}$ halts and returns 1
Return $\textsc{Dec}(c)$

$\underline{\mathcal{B}_E(pp : \textsc{SKeyGen}(\cdot), \textsc{SimSign}(\cdot,\cdot,\cdot))}$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{pke}}(1^\lambda)$
For $i = 1, \ldots, n$
$sk_i \leftarrow\!\!\$\ \textsc{SKeyGen}((ek, i))$
$\boldsymbol{gsk}[i] \leftarrow (pp, ek, i, sk_i)$
$b' \leftarrow\!\!\$\ \mathcal{A}((pp, ek), \boldsymbol{gsk} : \mathrm{LR}_{\mathcal{B}_E}(\cdot,\cdot,\cdot), \textsc{Open}_{\mathcal{B}_E}(\cdot,\cdot))$
Return b'

$\underline{\mathrm{LR}_{\mathcal{B}_E}(i_0, i_1, m)}$
$r^* \leftarrow\!\!\$\ \{0,1\}^\lambda$
$c^* \leftarrow \mathsf{Enc}(ek, 0; r^*)$
$\sigma^* \leftarrow\!\!\$\ \textsc{SimSign}((c^*, m))$
Return (c^*, σ^*)

$\underline{\textsc{Open}_{\mathcal{B}_E}(m, (c, \sigma))}$
If $\mathsf{Verify}(pp, (c, m), \sigma) = 0$ then return $\bot$
If $c = c^*$ and $\sigma \neq \sigma^*$ (QRY)
$\mathcal{B}_E$ halts and returns $((c^*, m), \sigma)$
Return $\textsc{Dec}(c)$

Figure 12: Adversaries $\mathcal{B}^{(b)}$ against IND-CCA of $\mathcal{PKE}$ and $\mathcal{B}_E$ against EXT of $\mathcal{PBS}$

We show that by IND-CCA of $\mathcal{PKE}$ and EXT of $\mathcal{PBS}$, $\mathbf{Exp}^{\text{anon-(1)-}b}$ and $\mathbf{Exp}^{\text{anon-(2)}}$ are indistinguishable. In Figure 12 we define an adversary $\mathcal{B}^{(b)}$ against $\mathcal{PKE}$ which uses an adversary $\mathcal{A}$ that behaves differently in these two games to break IND-CCA of $\mathcal{PKE}$.

We first define the event QRY marked in the description of $\mathcal{B}^{(b)}$: QRY_b denotes the event that in $\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}[\mathcal{PKE},\mathcal{PBS}],\mathcal{A}}$, $\mathcal{A}$ makes a valid $\textsc{Open}$ query $(m, (c^*, \sigma))$ with $\sigma \neq \sigma^*$. QRY_z denotes the event that $\mathcal{A}$ does so in $\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS}[\mathcal{PKE},\mathcal{PBS}],\mathcal{A}}$, i.e., when c^* is an encryption of 0. In our analysis, we require the following lemma, which we prove below.

Lemma 1. *The probability that the event* QRY_z *happens is upper-bounded by* $\mathbf{Adv}^{\text{EXT}}_{\mathcal{PBS},\mathcal{B}_E[\mathcal{A}]}$ *with* $\mathcal{B}_E[\mathcal{A}]$ *defined in Figure 12.*

We now show that if $\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{PKE},\mathcal{B}^{(b)}[\mathcal{A}]}$ and $\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS},\mathcal{B}_E[\mathcal{A}]}$ are negligible then so is $\mathbf{Adv}^{\text{anon-(1)}}_{\mathcal{GS}[\mathcal{PKE},\mathcal{PBS}],\mathcal{A}}$, which implies anonymity of our group-signature scheme. Analyzing the behavior of $\mathcal{B}^{(b)}$ in $\mathbf{Exp}^{\text{IND-CCA-}d}_{\mathcal{PKE}}$, for $d = 0, 1$, we get:

$$\begin{aligned}\Pr[\mathbf{Exp}^{\text{IND-CCA-1}}_{\mathcal{PKE},\mathcal{B}^{(b)}[\mathcal{A}]} = 1] &= \Pr[\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS},\mathcal{A}} = 1 \wedge \neg\mathsf{QRY}_b] + \Pr[\mathsf{QRY}_b] \\ &\geq \Pr[\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS},\mathcal{A}} = 1]\end{aligned}$$

and

$$\begin{aligned}\Pr[\mathbf{Exp}^{\text{IND-CCA-0}}_{\mathcal{PKE},\mathcal{B}^{(b)}[\mathcal{A}]} = 1] &= \Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}} = 1 \wedge \neg\mathsf{QRY}_z] + \Pr[\mathsf{QRY}_z] \\ &\leq \Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}} = 1] + \mathbf{Adv}^{\text{EXT}}_{\mathcal{PBS},\mathcal{B}_E[\mathcal{A}]}\end{aligned}$$

Together this yields

$$\Pr[\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS},\mathcal{A}} = 1] - \Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}} = 1] \leq \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{PKE},\mathcal{B}^{(b)}[\mathcal{A}]} + \mathbf{Adv}^{\text{EXT}}_{\mathcal{PBS},\mathcal{B}_E[\mathcal{A}]} \tag{8}$$

It remains now to lower-bound the probability on the left-hand side of (8). Let $\overline{\mathcal{B}}^{(b)}$ be defined as $\mathcal{B}^{(b)}$ but behaving like $\mathcal{B}^{(b)}$ does in $\mathbf{Exp}^{\text{IND-CPA-}(1-b)}$, that is: when answering $\mathcal{A}$'s $\mathrm{LR}_{\mathcal{B}}$ query, it queries its own LR oracle on $(i_b, 0)$; moreover, if during an $\textsc{Open}$ query the event QRY happens, it returns 0 (rather than

1). We then get:

$$\begin{aligned}\Pr[\mathbf{Exp}^{\text{IND-CCA-1}}_{\mathcal{PKE},\overline{\mathcal{B}}^{(b)}[\mathcal{A}]}=1] &= \Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}}=1\wedge\neg\mathsf{QRY}_z]\\ &= \Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}}=1]-\Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}}=1\wedge\mathsf{QRY}_z]\\ &\geq \Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}}=1]-\Pr[\mathsf{QRY}_z]\end{aligned}$$

and

$$\begin{aligned}\Pr[\mathbf{Exp}^{\text{IND-CCA-0}}_{\mathcal{PKE},\overline{\mathcal{B}}^{(b)}[\mathcal{A}]}=1] &= \Pr[\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS},\mathcal{A}}=1\wedge\neg\mathsf{QRY}_b]\\ &\leq \Pr[\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS},\mathcal{A}}=1]\end{aligned}$$

Together this yields

$$\Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}}=1]-\Pr[\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS},\mathcal{A}}=1] \;\leq\; \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{PKE},\overline{\mathcal{B}}^{(b)}[\mathcal{A}]}+\mathbf{Adv}^{\text{EXT}}_{\mathcal{PBS},\mathcal{B}_E[\mathcal{A}]}\;. \tag{9}$$

(8) and (9) together yield:

$$\big|\Pr[\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS},\mathcal{A}}=1]-\Pr[\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS},\mathcal{A}}=1]\big| \leq \max\big\{\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{PKE},\mathcal{B}^{(b)}[\mathcal{A}]},\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{PKE},\overline{\mathcal{B}}^{(b)}[\mathcal{A}]}\big\}+\mathbf{Adv}^{\text{EXT}}_{\mathcal{PBS},\mathcal{B}_E[\mathcal{A}]}\;,$$

thus, assuming $\mathcal{PKE}$ satisfies IND-CCA and $\mathcal{PBS}$ satisfies EXT, the games $\mathbf{Exp}^{\text{anon-(1)-}b}_{\mathcal{GS}}$ and $\mathbf{Exp}^{\text{anon-(2)}}_{\mathcal{GS}}$ are indistinguishable. Since the last game is independent of the bit b, together we have that games $\mathbf{Exp}^{\text{anon-1}}_{\mathcal{GS}[\mathcal{PKE},\mathcal{PBS}],\mathcal{A}}$ and $\mathbf{Exp}^{\text{anon-0}}_{\mathcal{GS}[\mathcal{PKE},\mathcal{PBS}],\mathcal{A}}$ are indistinguishable. It remains to prove Lemma 1.

Proof of Lemma 1. We construct an adversary $\mathcal{B}_E$ which breaks EXT whenever QRY happens in the game where $\mathcal{A}$ is given as c^* an encryption of 0. $\mathcal{B}_E$ is given on the right of Figure 12 and perfectly simulates $\mathbf{Exp}^{\text{IND-CCA-0}}_{\mathcal{PKE},\mathcal{B}[\mathcal{A}]}$.

We show that whenever the event QRY happens then $\mathcal{B}_E$ wins $\mathbf{Exp}^{\text{EXT}}_{\mathcal{PBS}}$ by returning $((c^*,\hat{m}),\hat{\sigma})$: We have to show that in this case we have (1) $\mathsf{Verify}(pp,(c^*,\hat{m}),\hat{\sigma})=1$; (2) $((c^*,\hat{m}),\hat{\sigma})$ is not among the query/response pairs for SimSign calls; and (3) with $((\hat{ek},\hat{\imath}),\hat{r})\leftarrow\mathsf{Extr}(tr,(c^*,\hat{m}),\hat{\sigma})$ we have either (3a) $(\hat{ek},\hat{\imath})$ was never queried to SKeyGen or (3b) $\mathsf{PC}\big(((\hat{ek},\hat{\imath}),(c^*,\hat{m})),\hat{r}\big)=0$.

(1) is the case as otherwise $\mathcal{B}_E$ would not have halted in the $\textsc{Open}_{\mathcal{B}}$ query. (2) is satisfied, as σ^* is the only reply of a SimSign call, and by (QRY) we have $\hat{\sigma}\neq\sigma^*$. Suppose (3a) is not satisfied, which means $\hat{ek}$ is the encryption key set up by $\mathcal{B}_E$ and $1\leq\hat{\imath}\leq n$. By Equation (4), Condition (3b) means $c^*\neq\mathsf{Enc}(ek,\hat{\imath};\hat{r})$. This is satisfied, since $c^*=\mathsf{Enc}(ek,0;r^*)$ and by correctness of $\mathcal{PKE}$, c^* cannot be the encryption of a different message. □

F Details of Primitives Implied by PBS

F.1 Definition of Simulation-Extractable NIZKs

Simulation-extractable NIZK (SE-NIZK) have been introduced and formalized as *simulation-sound extractable proofs* in the full version of [Gro06]. (We simplify slightly here, in that we do not distinguish between the trapdoors for simulation and extraction, and thus only have one $\mathsf{SimSetup}$ algorithm.) A SE-NIZK scheme $\mathcal{NIZK}$ for an **NP**-language defined by relation R consists of the following algorithms: Setup takes a security parameter 1^λ and outputs a common reference string *crs*; $\mathsf{SimSetup}$ additionally outputs a trapdoor *tr*. Prove takes as input *crs*, a statement $x\in\{0,1\}^*$ and a witness $w\in\{0,1\}^*$ and outputs a proof π. $\mathsf{SimProve}$ outputs a proof on input the trapdoor *tr* and a statement x. Verify, on input crs,x,π outputs a bit and Extr, on input tr,x,π outputs a witness w. A scheme $\mathcal{NIZK}$ is *correct* relative to an **NP**-relation R if for all $\lambda\in\mathbb{N}$, all $x,w\in\{0,1\}^*$ so that $\mathsf{R}(x,w)=1$, all $crs\in[\mathsf{Setup}(1^\lambda)]$ and all $\pi\in[\mathsf{Prove}(crs,x,w)]$ we have $\mathsf{Verify}(crs,x,\pi)=1$.

$\mathbf{Exp}^{\mathrm{ZK}}_{\mathcal{NIZK}}$

proc INITIALIZE
$b \leftarrow \{0,1\}$
$(crs_0, tr) \leftarrow \mathsf{SimSetup}(1^\lambda)$
$(crs_1) \leftarrow \mathsf{Setup}(1^\lambda)$
Return crs_b

proc PROOF(x, w)
If $\mathsf{R}(x,w) = 1$ then $\pi_0 \leftarrow\!\!\$\ \mathsf{SimProve}(tr, x)$
else $\pi_0 \leftarrow \bot$
$\pi_1 \leftarrow\!\!\$\ \mathsf{Prove}(crs_1, x, w)$
Return π_b

proc FINALIZE(b')
Return $(b = b')$

$\mathbf{Exp}^{\mathrm{SE}}_{\mathcal{NIZK}}$

proc INITIALIZE
$(crs, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}(1^\lambda)\ ;\ Q \leftarrow \emptyset$
return crs

proc SIMPROVE(x)
$\pi \leftarrow\!\!\$\ \mathsf{SimProve}(crs, tr, x)\ ;\ Q \leftarrow Q \cup \{(x, \pi)\}$
return π

proc FINALIZE(x, π)
$w \leftarrow \mathsf{Extr}(tr, x, \pi)$
Return 1 if all of the following hold:
$(x, \pi) \notin Q$
$\mathsf{Verify}(crs, x, \pi) = 1$
$\mathsf{R}(x, w) = 0$

Figure 13: Games defining zero-knowledge and simulation-extractability for NIZK

Security is defined via the two experiments $\mathbf{Exp}^{\mathrm{ZK}}_{\mathcal{NIZK}}$ and $\mathbf{Exp}^{\mathrm{SE}}_{\mathcal{NIZK}}$ in Figure 13. A proof system $\mathcal{NIZK}$ = $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimSetup}, \mathsf{SimProve}, \mathsf{Extr})$ for a relation R is *zero-knowledge* if $\mathbf{Adv}^{\mathrm{ZK}}_{\mathcal{NIZK},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{ZK}}_{\mathcal{NIZK},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}] - \frac{1}{2}$ is negligible in λ for all PT $\mathcal{A}$. It is *simulation-extractable* if $\mathbf{Adv}^{\mathrm{SE}}_{\mathcal{NIZK},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{SE}}_{\mathcal{NIZK},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}]$ is negligible in λ for all PT $\mathcal{A}$.

F.2 Details on Public-Key Encryption from PBS

If $\mathcal{PBS}$ satisfies IND then it follows immediately that $\mathcal{PKE} = (\mathsf{KeyGen}_{\mathrm{pke}}, \mathsf{Enc}, \mathsf{Dec})$, as defined in Section 5.2, satisfies indistinguishability of ciphertexts under chosen-plaintext attack (IND-CPA).

Sahai [Sah99] shows that when instantiating the Naor-Yung [NY90] construction with a simulation-sound NIZK, one obtains a chosen-ciphertext-attack (CCA)-secure public-key encryption scheme, which is what we require for our construction of group signatures in Section 5.1. Combining our constructions of SE NIZK and CPA PKE from Section 5.2, we thus obtain a CCA-secure PKE; which we could also directly construct as follows:

Let $\mathcal{PKE}_{\mathrm{cpa}} = (\mathsf{KeyGen}_{\mathrm{cpa}}, \mathsf{Enc}_{\mathrm{cpa}}, \mathsf{Dec}_{\mathrm{cpa}})$ be a CPA-secure PKE; let $\mathcal{PBS}$ be a policy-based signature scheme for the policy checker

$$\mathsf{PC}(((pk_0, pk_1), (c_0, c_1)), (x, r_0, r_1)) = 1 \iff c_0 = \mathsf{Enc}_{\mathrm{pke}}(pk_0, x; r_0) \ \wedge\ c_1 = \mathsf{Enc}_{\mathrm{pke}}(pk_1, x; r_1) \qquad (10)$$

Then the following scheme $\mathcal{PKE}_{\mathrm{cca}}$ is an IND-CCA secure PKE scheme:

$\underline{\mathsf{KeyGen}_{\mathrm{cca}}(1^\lambda)}$
$(pk_0, dk_0) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{cpa}}(1^\lambda)$
$(pk_1, dk_1) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{cpa}}(1^\lambda)$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}_{\mathrm{pbs}}(1^\lambda)$
$sk \leftarrow\!\!\$\ \mathsf{KeyGen}(msk, (pk_0, pk_1))$
Return $\overline{pk} \leftarrow (pk_0, pk_1, sk)$
$\overline{dk} \leftarrow (pp, dk_0)$

$\underline{\mathsf{Enc}_{\mathrm{cca}}((pk_0, pk_1, sk), x)}$
$r_0, r_1 \leftarrow\!\!\$\ \{0,1\}^\lambda$
$c_0 \leftarrow \mathsf{Enc}_{\mathrm{cpa}}(pk_0, x; r_0)$
$c_1 \leftarrow \mathsf{Enc}_{\mathrm{cpa}}(pk_1, x; r_1)$
$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, (c_0, c_1), (x, r_0, r_1))$
Return $\overline{c} = (c_0, c_1, \sigma)$

$\underline{\mathsf{Dec}_{\mathrm{cca}}((pp, dk_0), (c_0, c_1, \sigma))}$
If $\mathsf{Verify}(pp, (c_0, c_1), \sigma) = 0$
Return $\bot$
Return $\mathsf{Dec}_{\mathrm{cpa}}(dk_0, c_0)$

F.3 Efficient Constructions

Group signatures. The efficient construction of policy-based signatures given in Section 4.2 requires that the policy checker PC be expressible as a set of pairing-product equations. For our construction of group signatures from PBS in Section 5.1, this means that Equation (4) must be expressed as an equation of the form given in Equation (2).

We thus need to express the predicate "*is an encryption of my identity*" as a statement in the language of pairing-product equations. The witness therefore must be a group element, so it seems that we need an

encryption scheme whose randomness is a group element. It is however sufficient to find a group element which witnesses the fact that a ciphertext is the encryption of a certain plaintext.

Let user identities be elements $I \in \mathbb{G}$ and define the opener's public key as $Y \in \mathbb{G}$. An ElGamal encryption of I under public key Y is defined by choosing $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and setting $(C, D) \leftarrow (I \cdot Y^r, G^r)$. Using the bilinear map, $W \leftarrow H^r$ is a witness for encryption of I using the two equations:

$$e(C, H) = e(I, H)\, e(Y, \underline{W}) \qquad\qquad e(D, H) = e(G, \underline{W}) \tag{11}$$

Thus, for a policy I (which we identify with the user identity) and a message of the form (C, D, M), the policy checker for our group-signature construction is defined as

$$\mathsf{PC} := \{((I, (C, D, M)), W) \mid (I, C, D, W) \text{ satisfy Equation (11)}\} \ .$$

In order to efficiently instantiate our group-signature construction given in Section 5.1, we thus require a CCA-secure encryption, which contains as part of a ciphertext an ElGamal ciphertext, which we show how to construct next.

CCA-secure encryption. In Section 5.2, we showed how to combine two ciphertexts and a policy-based signature on them to a CCA-secure public-key encryption. The policy checker for the PBS is defined in Equation (10). Using as the CPA-secure scheme ElGamal encryption, this can be expressed as a set of pairing-product equations, where $Y_1, Y_2 \in \mathbb{G}$ are be the public keys defining the policy, the ciphertexts $(C_0, D_0) \in \mathbb{G}^2$ and $(C_1, D_1) \in \mathbb{G}^2$ represent the PBS-message and $(X, W_0, W_1) \in \mathbb{G} \times \mathbb{H}^2$ is the witness:

$$\begin{aligned} e(C_0, H) &= e(\underline{X}, H)\, e(Y_0, \underline{W}_0) & \qquad e(D_0, H) &= e(G, \underline{W}_0) \\ e(C_1, H) &= e(\underline{X}, H)\, e(Y_1, \underline{W}_1) & \qquad e(D_1, H) &= e(G, \underline{W}_1) \end{aligned}$$

G Details of Delegatable Policy-Based Signatures

G.1 Unforgeability and Indistinguishability

Unforgeability is defined similarly to the non-delegatable case. We let the adversary produce keys, without seeing them, using a DELEGATE oracle, allowing it to derive keys from other keys. The adversary can then either ask for a key to be revealed, using his REVEALSK oracle, or for signatures on messages of his choice under keys of his choice. The adversary breaks the notion if he outputs a valid message/signature pair (m, σ) which he could not have produced trivially. That is, if in the list of created keys, there is a key corresponding to policies which the message all satisfies then the adversary must not have queried REVEALSK on it and m must not be in the list of messages which were queried to Sign under that key.

We formalize this using three oracles $\mathsf{Delegate}$, RevSK and Sign, which maintain a list Q with entries of the form $(p_1 \| \ldots \| p_k, sk, \{m_1, \ldots, m_n\})$. Such an entry being in Q means that sk is a secret key associated to policies $p_1, \ldots, p_k$ which has been used to sign messages $m_1, \ldots, m_n$. We initialize the list with $Q[0] := (\emptyset, msk, \emptyset)$. With $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{DPBS}}$ defined in Figure 14, we say that $\mathcal{DPBS}$ is UF-secure if $\Pr[\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{DPBS}} \Rightarrow \mathsf{true}]$ is negligible in λ for every PT $\mathcal{A}$.

The difference in the indistinguishability experiment with respect to PBS is that the adversary can itself compute two secret keys sk_0 and sk_1 (since it obtains msk) for policy vectors $\mathbf{p}_1$ and $\mathbf{p}_2$ and then ask for a signature under one of them. We assume that the adversary outputs the two policy vectors of equal length (as for PBS, we do not require that the *length* of policies is hidden). In order to prevent the adversary from trivially winning by outputting malfunctioning keys, the experiment checks them using ChkKey. With $\mathbf{Exp}^{\mathrm{IND}}_{\mathcal{DPBS}}$ defined in Figure 14, we say that $\mathcal{DPBS}$ is IND-secure if $\mathbf{Adv}^{\mathrm{IND}}_{\mathcal{DPBS}} = \Pr[\mathbf{Exp}^{\mathrm{IND}}_{\mathcal{DPBS}} \Rightarrow \mathsf{true}] - \frac{1}{2}$ is negligible in λ for every PT $\mathcal{A}$.

Analogously to the proof in Appendix A, one can show that simulatability and extractability imply unforgeability and indistinguishability.

$\mathbf{Exp}^{\text{UF}}_{\mathcal{DPBS}}$

proc INITIALIZE
$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$; $Q[0][2] \leftarrow msk$; $j \leftarrow 0$
Return pp

proc DELEGATE(i, p')
If $i \notin \{0 \dots j\}$ or $Q[i][2] = \bot$ then return $\bot$
$j \leftarrow j + 1$; $\mathbf{p} \leftarrow Q[i][1]$; $sk \leftarrow Q[i][2]$
$sk' \leftarrow \mathsf{Delegate}(sk, p')$; $Q[j] \leftarrow (\mathbf{p} \| p', sk', \emptyset)$

proc REVEALSK(i)
If $i \notin [j]$ then return $\bot$
$sk \leftarrow Q[i][2]$; $Q[i][2] \leftarrow \bot$; return sk

proc SIGN(i, m, w)
If $i \notin [j]$ or $Q[i][2] = \bot$ then return $\bot$
$Q[i][3] \leftarrow Q[i][3] \cup \{m\}$
Return $\mathsf{Sign}(pp, Q[i][2], m, w)$

proc FINALIZE(m, σ)
If $\mathsf{Verify}(pp, m, \sigma) = 0$ then return false
For $i = 1, \dots, j$ do
  If $(Q[i][1], m) \in \mathcal{L}(\mathsf{PC})$ then
    If $Q[i][2] = \bot$ or $m \in Q[i][3]$
      then return false
Return true

$\mathbf{Exp}^{\text{IND}}_{\mathcal{DPBS}}$

proc INITIALIZE
$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$; $b \leftarrow\!\!\$\ \{0, 1\}$
Return (pp, msk)

proc LR$(sk_0, \mathbf{p}_0, sk_1, \mathbf{p}_1, m, \mathbf{w}_0, \mathbf{w}_1)$
If $\mathsf{ChkKey}(pp, sk_0, \mathbf{p}_0) = 0$ or $\mathsf{ChkKey}(pp, sk_1, \mathbf{p}_1) = 0$
  then return $\bot$
If $\mathsf{PC}((\mathbf{p}_0, m), \mathbf{w}_0) = 0$ or $\mathsf{PC}((\mathbf{p}_1, m), \mathbf{w}_1) = 0$
  then return $\bot$
$\sigma_b \leftarrow \mathsf{Sign}(sk_b, m, \mathbf{w}_b)$
Return σ_b

proc FINALIZE(b')
Return $(b = b')$

Figure 14: Games defining unforgeability and indistinguishability for delegatable PBS.

G.2 Append-Only Signatures

Definition. We let ε denote a sequence of messages of length 0. An append-only signature (AOS) consists of three algorithms: KeyGen takes 1^λ as input and outputs a key pair (vk, sk); on input a verification key vk, a signature σ or a secret key sk, and a message m', Append outputs a signature σ'. On input $vk, (m_1, \dots, m_n), \sigma$, Verify outputs 0 or 1. *Correctness* requires that for all $(vk, sk) \in [\mathsf{KeyGen}(1^\lambda)]$, every sequence of message $(m_1, \dots, m_1)$, and all $\sigma \in [\mathsf{Append}(vk, \mathsf{Append}(vk, \dots, \mathsf{Append}(vk, \mathsf{Append}(vk, sk, m_1), m_2), \dots, m_{n-1}), m_n)]$, we have $\mathsf{Verify}[vk, (m_1, \dots, m_n), \sigma) = 1$.

An AOS scheme $\mathcal{AOS}ig$ should be unforgeable against chosen-message attacks, of which we give a more general definition than [KMPR05]: We require that $\Pr[\mathbf{Exp}^{\text{UF}}_{\mathcal{AOS}ig, \mathcal{A}} \Rightarrow \mathsf{true}]$ is negligible for any PT $\mathcal{A}$, with $\mathbf{Exp}^{\text{UF}}_{\mathcal{AOS}ig}$ defined in Figure 15.

Instantiation. We now argue that the generic construction of AOS given by Kiltz et al. [KMPR05] satisfies our unforgeability definition. The construction is based on any existentially unforgeable signature scheme

$\mathbf{Exp}^{\text{UF}}_{\mathcal{AOS}ig}$

proc INITIALIZE
$(vk, sk) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$
$j \leftarrow 0$; $Q[0] \leftarrow (\varepsilon, sk)$; $Q_M \leftarrow \emptyset$; return vk

proc FINALIZE$(((m_1^*, \dots, m_n^*), \sigma^*))$
If $\mathsf{Verify}(vk, (m_1^*, \dots, m_n^*), \sigma^*) = 0$
  then return 0
If for any $\mathbf{m} \in Q_M$ is a prefix of $(m_1^*, \dots, m_n^*)$
  then return 0
Return 1

proc APPEND(i, m')
$j \leftarrow j + 1$
$Q[j][1] \leftarrow Q[i][1] \| m'$
$Q[j][2] \leftarrow\!\!\$\ \mathsf{Append}(vk, Q[i][2], m')$

proc REVEALSIG(i)
$\sigma \leftarrow Q[i][2]$; $Q_M \leftarrow Q_M \cup \{Q[i][1]\}$
Return σ

Figure 15: Game defining unforgeability for append-only signatures

$\mathcal{S}ig = (\mathsf{KeyGen}_{\mathrm{sig}}, \mathsf{Sign}_{\mathrm{sig}}, \mathsf{Verify}_{\mathrm{sig}})$: KeyGen is defined as $\mathsf{KeyGen}_{\mathrm{sig}}$ and Append and Verify are defined as follows:

$\underline{\mathsf{Append}(vk, \sigma, m')}$
If σ is a $\mathcal{S}ig$ secret key, set $sk_0 \leftarrow \sigma$
Else parse σ as $(\sigma_1, vk_1, \ldots, \sigma_n, vk_n, sk_n)$
$(vk_{n+1}, sk_{n+1}) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{sig}}(1^\lambda)$
$\sigma_{n+1} \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{sig}}(sk_n, (m', vk_{n+1}))$
Return $(\sigma_1, vk_1, \ldots, \sigma_n, vk_n, \sigma_{n+1}, vk_{n+1}, sk_{n+1})$

$\underline{\mathsf{Verify}(vk_0, (m_1, \ldots, m_n), \sigma)}$
Parse σ as $(\sigma_1, vk_1, \ldots, \sigma_n, vk_n, sk_n)$
For $i = 1, \ldots, n$:
  If $\mathsf{Verify}_{\mathrm{sig}}(vk_{i-1}, (m_i, vk_i), \sigma_i) = 0$, return 0
For a fixed message $\hat{m}$
Return $\mathsf{Verify}_{\mathrm{sig}}(vk_n, m, \mathsf{Sign}_{\mathrm{sig}}(sk_n, m))$

From an adversary winning $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{AOS}ig}$ with probability ϵ, one can construct an adversary $\mathcal{B}$ against EUF-CMA of $\mathcal{S}ig$ that wins with probability $\frac{\epsilon}{q_{\max}+1}$, where $q_{\max}$ is an upper bound on the number of APPEND queries that $\mathcal{A}$ makes.

$\mathcal{B}$ receives a $\mathcal{S}ig$ key $\widehat{vk}$ from its challenger and guesses $\widehat{\jmath} \leftarrow\!\!\$\ [0, q_{\max}]$. If $\widehat{\jmath} = 0$, it sets $vk_0 \leftarrow \widehat{vk}$, otherwise it computes $(vk_0, sk_0) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{sig}}(1^\lambda)$. Then $\mathcal{B}$ runs $\mathcal{A}$ on input vk_0. $\mathcal{B}$ maintains a list Q as in $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{AOS}ig}$, which it initializes with $Q[0] \leftarrow (\varepsilon, sk_0)$ or $(\varepsilon, \perp)$ if $\widehat{\jmath} = 0$. For the $\widehat{\jmath}$-th APPEND query (that is, when $j = \widehat{\jmath}$) for (i, m'), if $Q[i][2] = (\sigma_1, vk_1, \ldots, \sigma_n, vk_n, sk_n)$, $\mathcal{B}$ sets $\sigma_{n+1} \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{sig}}(sk_n, (m', \widehat{vk}))$ and sets $Q[\widehat{\jmath}][2] = (\sigma_1, vk_1, \ldots, \sigma_n, vk_n, \sigma_{n+1}, \widehat{vk}, \perp)$ (thus using $\widehat{vk}$ instead of creating a new key pair). Whenever afterwards $\mathcal{A}$ queries $\textsc{Append}(\widehat{\jmath}, m'')$, $\mathcal{B}$ creates a new key (vk_{n+2}, sk_{n+2}) and uses its signing oracle to get a signature σ_{n+2} on (m'', vk_{n+2}) under $vk_{n+1} = \widehat{vk}$. All other APPEND queries are answered normally. If $\mathcal{A}$ ever queries $\textsc{RevealSig}(\widehat{\jmath})$, $\mathcal{B}$ aborts. It is easily seen that as long as it does not abort, $\mathcal{B}$ correctly simulates $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{AOS}ig,\mathcal{A}}$.

Let $(m^* = (m^*_1, \ldots, m^*_{n^*}), \sigma^* = (\sigma^*_1, vk^*_1, \ldots, \sigma^*_{n^*}, vk^*_{n^*}, sk^*_{n^*}))$ be a successful output by $\mathcal{A}$. $\mathcal{B}$ looks for the entry in Q that is the longest prefix of messages and verification keys of (m^*, σ^*); in particular, let j^* be such that $Q[j^*][1] = (m^*_1, \ldots, m^*_\ell)$, with $\ell \le n^*$, and $Q[j^*][2] = (\sigma'_1, vk^*_1, \ldots, \sigma'_\ell, vk^*_\ell, sk'_\ell)$, for any $(\sigma'_1, \ldots, \sigma'_\ell, sk'_\ell)$, so that ℓ is maximal. If no such entry exist then set $j^* = 0$ and $\ell = 0$.

We claim that if $\mathcal{B}$ guessed $\widehat{\jmath} = j^*$ then it breaks unforgeability. When the guess was correct then $vk^*_\ell = \widehat{vk}$ (and $sk_\ell = \perp$). If $\ell = n^*$ then $\mathcal{B}$ computes $\hat{\sigma} \leftarrow\!\!\$\ \mathsf{Sign}_{\mathrm{sig}}(sk^*_{n^*}, \hat{m})$ and outputs $(\hat{m}, \hat{\sigma})$, which is a valid pair with at least the probability that $\mathcal{A}$ wins $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{AOS}ig}$. Otherwise, it outputs the message $(m^*_{\ell+1}, vk^*_{\ell+1})$ and the signature $\sigma^*_{\ell+1}$ which verifies under $vk^*_\ell = \widehat{vk}$, since σ^* is valid. Moreover, $\mathcal{B}$ has never queried the message $(m^*_{\ell+1}, vk^*_{\ell+1})$ to its signing oracle, since otherwise ℓ would not be maximal. Finally, $\mathcal{B}$ did not abort the simulation, since $\mathcal{A}$ cannot have queried $\textsc{RevealSK}(\widehat{\jmath})$ because $Q[\widehat{\jmath}][1] = (m^*_1, \ldots, m^*_\ell)$ is a prefix of m^*.

G.3 Proof of Theorem 4

In Figure 16 we have rewritten the simulatability and extractability experiments from Figure 6 substituting the code of our construction. (We have simplified $\mathbf{Exp}^{\mathrm{EXT}}_{\mathcal{DPBS}[]}$, since entries in $Q[i][2]$ are computed by the experiment and are of the form (pp, p, c); thus $Q[i][1] = p$, meaning the entries in $Q[i][1]$ are redundant.)

Simulatability. We reduce simulatability to zero-knowledge of $\mathcal{NIZK}$ (see Figure 13 in Appendix F.1). Let $\mathcal{A}$ be an adversary for $\mathbf{Exp}^{\mathrm{SIM}}_{\mathcal{DPBS}}$; we construct an adversary $\mathcal{B}$ for $\mathbf{Exp}^{\mathrm{ZK}}_{\mathcal{NIZK}}$, which wins with the same probability. $\mathcal{B}$ receives crs_* from its challenger and computes $(mvk, msk') \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{sig}}(1^\lambda)$ and runs $\mathcal{A}$ on input $pp \leftarrow (crs_*, mvk), msk \leftarrow (pp, \varepsilon, msk')$. This simulates INITIALIZE of $\mathbf{Exp}^{\mathrm{SIM}}_{\mathcal{DPBS}}$ with the bit b set to $\mathcal{B}$'s challenger's bit.

Whenever $\mathcal{A}$ queries $\textsc{Signature}(sk, \mathbf{p}, m, \mathbf{w})$, $\mathcal{B}$ verifies the conditions in the first line of the experiment and returns $\perp$ if any fails. Otherwise $\mathcal{B}$ queries $\textsc{Proof}((mvk, m), (\mathbf{p}, c, \mathbf{w}))$ to receive σ_*. Note that since $\mathsf{Verify}_{\mathrm{sig}}(mvk, \mathbf{p}, c) = 1$, we have $R_{\mathrm{NP}}((vk, m), (\mathbf{p}, c, \mathbf{w}))$ iff $\mathsf{PC}((\mathbf{p}, m), \mathbf{w}) = 1$. Thus, $\mathcal{B}$ simulates σ_b where b is the bit of its own challenger. When $\mathcal{A}$ stops and outputs b', $\mathcal{B}$ forwards this to its own challenger. $\mathcal{B}$ thus guesses the bit correctly whenever $\mathcal{A}$ does.

Extractability. We distinguish between two types of adversaries winning $\mathbf{Exp}^{\mathrm{EXT}}_{\mathcal{DPBS}[]}$: Let $(\mathbf{p}, c, \mathbf{w})$ be the output of $\mathsf{Extr}_{\mathrm{nizk}}$ in FINALIZE.

$\mathbf{Exp}^{\text{SIM}}_{\mathcal{DPBS}[]}$

proc INITIALIZE
$b \leftarrow\!\!\$\ \{0,1\}$
$(crs_0, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}_{\text{nizk}}(1^\lambda)$
$crs_1 \leftarrow\!\!\$\ \mathsf{Setup}_{\text{nizk}}(1^\lambda)$
$(mvk, msk') \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$
Return $(pp_b \leftarrow (crs_b, mvk), msk_b \leftarrow (pp_b, \varepsilon, msk'))$

proc SIGNATURE$((pp, \mathbf{q}, c), \mathbf{p}, m, \mathbf{w})$
If $pp \neq (crs_b, mvk)$ or $\mathbf{p} \neq \mathbf{q}$ or
  $\mathsf{Verify}_{\text{sig}}(mvk, \mathbf{p}, c) = 0$ then return $\perp$
If $\mathsf{PC}((\mathbf{p}, m), \mathbf{w}) = 1$ then
  $\sigma_0 \leftarrow\!\!\$\ \mathsf{SimProve}_{\text{nizk}}(crs_0, tr, (mvk, m))$
Else $\sigma_0 \leftarrow \perp$
$\sigma_1 \leftarrow\!\!\$\ \mathsf{Prove}_{\text{nizk}}(crs_1, (mvk, m), (\mathbf{q}, c, \mathbf{w}))$
Return σ_b

proc FINALIZE(b')
Return $(b = b')$

$\mathbf{Exp}^{\text{EXT}}_{\mathcal{DPBS}[]}$

proc INITIALIZE
$(crs, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}_{\text{nizk}}(1^\lambda)$
$(mvk, msk') \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$; $pp \leftarrow (crs, mvk)$
$Q[0][2] \leftarrow (pp, \varepsilon, msk')$; $Q_S, Q_P \leftarrow \emptyset$; $j \leftarrow 0$; return pp

proc DELEGATE(i, p')
If $i \notin \{0, \ldots, j\}$ or $Q[i][2] = \perp$ then return $\perp$
$j \leftarrow j + 1$; $(pp = (crs, mvk), \mathbf{p}, c) \leftarrow Q[i][2]$
$c' \leftarrow\!\!\$\ \mathsf{Append}_{\text{sig}}(mvk, c, p')$; $Q[j][2] \leftarrow (pp, \mathbf{p}\|p', c')$

proc REVEALSK(i)
If $i \notin [j]$ then return $\perp$
$(pp, \mathbf{p}, c) \leftarrow Q[i][2]$; $Q_P \leftarrow Q_P \cup \{\mathbf{p}\}$; return $(pp, \mathbf{p}, c)$

proc SIMSIGN(m)
$\sigma \leftarrow\!\!\$\ \mathsf{SimProve}_{\text{nizk}}(crs, tr, (mvk, m))$
$Q_S \leftarrow Q_S \cup \{(m, \sigma)\}$; return σ

proc FINALIZE(m, σ)
If $\mathsf{Verify}_{\text{nizk}}(crs, (mvk, m), \sigma) = 0$ or $(m, \sigma) \in Q_S$
  then return false
$(\mathbf{p}, c, \mathbf{w}) \leftarrow \mathsf{Extr}_{\text{nizk}}(tr, (mvk, m), \sigma)$
If no $\mathbf{q} \in Q_P$ is a prefix of $\mathbf{p}$, then return true
If $\mathsf{PC}((\mathbf{p}, m), \mathbf{w}) = 0$, then return true
Return false

Figure 16: Simulatability and Extractability for $\mathcal{DPBS}[\mathcal{AOSig}, \mathcal{NIZK}]$

- $\mathcal{A}$ is of Type 1 if $\mathsf{Verify}_{\text{sig}}(mvk, \mathbf{p}, c) = 1$ and no $\mathbf{q} \in Q_P$ is a prefix of $\mathbf{p}$.
- $\mathcal{A}$ is of Type 2 if $\mathsf{Verify}_{\text{sig}}(mvk, \mathbf{p}, c) = 0$ or $\mathsf{PC}((\mathbf{p}, m), \mathbf{w}) = 0$.

(Note that every winning adversary is of either Type 1 or Type 2.) We will use Type-1 adversaries to break unforgeability of $\mathcal{AOSig}$ and Type-2 adversaries to break simulation-extractability of $\mathcal{NIZK}$.

Let $\mathcal{A}$ be of Type 1. We construct $\mathcal{B}$ for $\mathbf{Exp}^{\text{UF}}_{\mathcal{AOSig}}$. $\mathcal{B}$ receives vk from its challenger, computes $(crs, tr) \leftarrow\!\!\$\ \mathsf{SimSetup}_{\text{nizk}}(1^\lambda)$, sets $Q_S \leftarrow \emptyset$ and runs $\mathcal{A}$ on $pp \leftarrow (crs, vk)$. A query SIMSIGN by $\mathcal{A}$ is dealt with honestly, since $\mathcal{B}$ holds tr; $\mathcal{B}$ also adds (m, σ) to Q_S. All DELEGATE queries are forwarded to $\mathcal{B}$'s APPEND oracle and REVEALSK queries by $\mathcal{A}$ are forwarded to $\mathcal{B}$'s own REVEALSIG oracle. Note that for any i: $Q[i][2] = (pp, \mathbf{p}, c)$ in $\mathbf{Exp}^{\text{EXT}}_{\mathcal{DPBS}[]}$ iff $(Q[i][1], Q[i][2]) = (\mathbf{p}, c)$ in $\mathbf{Exp}^{\text{UF}}_{\mathcal{AOSig}}$; and $\mathbf{p} \in Q_P$ in $\mathbf{Exp}^{\text{EXT}}_{\mathcal{DPBS}[]}$ iff $\mathbf{p} \in Q_M$ in $\mathbf{Exp}^{\text{UF}}_{\mathcal{AOSig}}$.

When $\mathcal{A}$ outputs (m, σ), $\mathcal{B}$ checks validity of σ and that the pair is not contained in Q_S and computes $(\mathbf{p}, c, \mathbf{w}) \leftarrow \mathsf{Extr}_{\text{nizk}}(tr, (mvk, m), \sigma)$. If $\mathcal{A}$ is of Type 1, we have $\mathsf{Verify}_{\text{sig}}(mvk, \mathbf{p}, c) = 1$ and for all $\mathbf{q} \in Q_P$: $\mathbf{q}$ is not a prefix of $\mathbf{p}$. $\mathcal{B}$ wins thus $\mathbf{Exp}^{\text{UF}}_{\mathcal{AOSig}}$ by outputting $(\mathbf{p}, \sigma)$.

Let $\mathcal{A}$ be of Type 2. We construct $\mathcal{B}$ for $\mathbf{Exp}^{\text{SE}}_{\mathcal{NIZK}}$. $\mathcal{B}$ receives crs from its challenger and runs $(mvk, msk') \leftarrow\!\!\$\ \mathsf{KeyGen}_{\text{sig}}(1^\lambda)$, and sets $Q[0][2] \leftarrow ((crs, mvk), \varepsilon, msk')$ and $Q_P \leftarrow \emptyset$. It then runs $\mathcal{A}$ on input $pp \leftarrow (crs, mvk)$. $\mathcal{B}$ answers DELEGATE and REVEALSK queries as specified, maintaining the lists Q and Q_P. Whenever $\mathcal{A}$ queries SIMSIGN(m), $\mathcal{B}$ queries SIMPROVE(mvk, m) and forwards the response to $\mathcal{A}$. $\mathcal{B}$ perfectly simulates $\mathbf{Exp}^{\text{EXT}}_{\mathcal{DPBS}[],\mathcal{A}}$ and $\mathcal{B}$'s challenger's list Q in $\mathbf{Exp}^{\text{SE}}_{\mathcal{NIZK}}$ corresponds to Q_S in $\mathbf{Exp}^{\text{EXT}}_{\mathcal{DPBS}[]}$. When $\mathcal{A}$ outputs (m, σ), $\mathcal{B}$ outputs $((mvk, m), \sigma)$.

If $\mathcal{A}$ is of Type 2 and wins, we have $\mathsf{Verify}_{\text{sig}}(mvk, \mathbf{p}, c) = 0$ or $\mathsf{PC}((\mathbf{p}, m), \mathbf{w}) = 0$, meaning that $R_{\text{NP}}((vk, m), (\mathbf{p}, c, \mathbf{w})) = 0$. Since we moreover have $\mathsf{Verify}_{\text{nizk}}(crs, (mvk, m), \sigma) = 1$ and $(m, \sigma) \notin Q_S$, $\mathcal{B}$'s output $((mvk, m), \sigma)$ satisfies the winning conditions in $\mathbf{Exp}^{\text{SE}}_{\mathcal{NIZK}}$.