Policy-Based Signatures

Mihir Bellare[1] and Georg Fuchsbauer[2]

[1] Department of Computer Science and Engineering, University of California San Diego, USA
[2] Institute of Science and Technology Austria

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.

1 Introduction

PBS. In a standard digital signature scheme [25, 29], 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 requires that signatures 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.

PBS may be viewed as an authentication analogue of functional encryption [15]. 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, 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 unifying existing notions by capturing some as special cases and allowing 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 [22, 10], proxy signatures [35], ring signatures [38, 14], mesh signatures [17], anonymous proxy signatures [28], attribute-based signatures [34] and anonymous credentials [19, 6].

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, simulatability-based privacy notion that says that real signatures look like ones a simulator could generate without knowledge of the policy or any key. 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 [15, 37] and stronger simulatability requirements have been defined and considered [15, 37, 11, 23, 2, 5, 36]. However, for FE, impossibility results show that the strongest and most desirable simulation-based definitions are not achievable [15, 11, 23, 2, 36]. In contrast, for PBS we 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 [30].

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 [12]. Combining Groth-Sahai proofs [31] and structure-preserving signatures [1], we 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 we show that PBS implies group signatures meeting the strong CCA version of the definition of [10]. In the full version [7] we also show that PBS implies attribute-based signatures [34] and signatures of knowledge [21]. 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 [7], we address this and

show that PBS implies seemingly unrelated primitives like IND-CPA encryption and simulation-extractable NIZK proofs [30]. By [39] 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. The holder of $sk_{p_1 \| p_2}$ can further delegate 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. For simplicity, we base our construction, achieving sim+ext security, on append-only signatures [33], 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 ABS or mesh signatures, the policy itself is always public, as is the warrant specifying 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 [8], 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 [10]. Our first generic construction builds on these ideas. Our second generic construction, inspired by [26, 9], 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) [18] 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 [20]. 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 [3] 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 [16, 18, 32]. 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 [16] also provide a policy-based key-distribution scheme.

In their treatment of policy-based cryptography, Bagga and Molva [4] 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. 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 and "PT" stands for "polynomial-time". 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 $x_1, \ldots$.

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 [13]. 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. The output of the execution, denoted $\mathbf{Exp}_\mathcal{A}(\lambda)$ is the output of FINALIZE. 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.

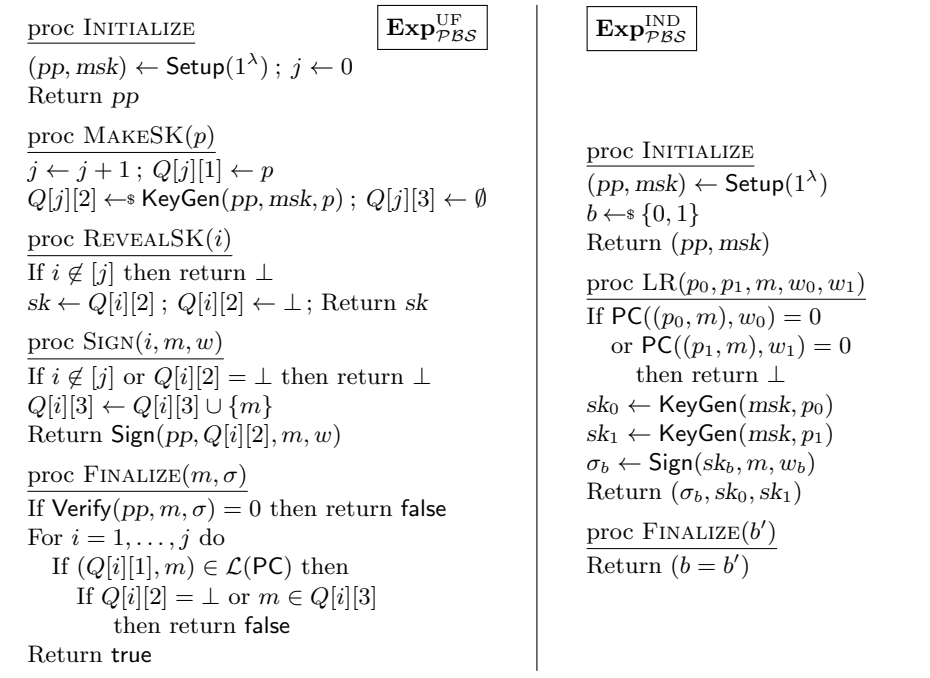

Fig. 1. Games defining unforgeability and indistinguishability for PBS.

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 (p_0, m, w_0) and (p_1, m, w_1) are PC-valid. For either p_0 or p_1 the experiment computes a secret key and uses it to produce a signature on m, from which the adversary has to determine 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 that $\mathcal{A}$ outputs 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 for an example of such a PC.) Now consider a scheme which satisfies indistinguishability, and modify it 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. 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 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 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 (4) in Section 5.) Although not a problem

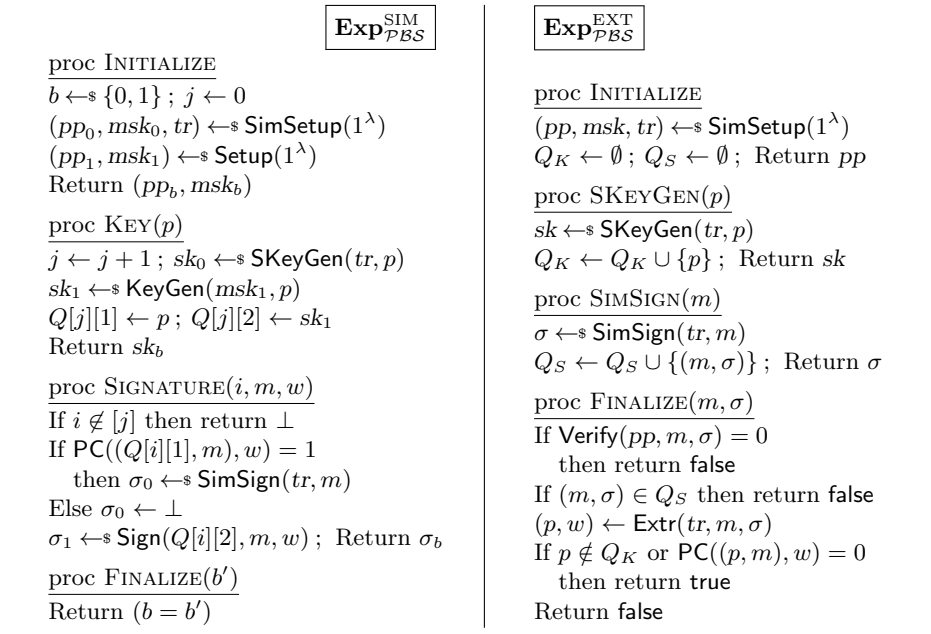

Fig. 2. Games defining simulatability and extractability for PBS

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 [7] for such a proof.) 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: SimSetup, which outputs parameters and a master key that are indistinguishable from those output by Setup, as well as a trapdoor; SKeyGen, which outputs keys indistinguishable from those output by KeyGen; and SimSign, which on input the trapdoor and a message (but no signing key nor witness) produces signatures that are indistinguishable from regular signatures.

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

Extractability. We define our notion in the spirit of *"sim-ext" security* for signatures of knowledge [21]. Let $\mathbf{Adv}^{\mathrm{EXT}}_{\mathcal{PBS},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{EXT}}_{\mathcal{PBS},\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}]$ with

$\mathbf{Exp}^{\mathrm{EXT}}_{\mathcal{PBS}}$ defined in 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 $\mathbf{Adv}^{\mathrm{EXT}}_{\mathcal{PBS},\mathcal{A}}(\cdot)$ 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 $\mathbf{Exp}^{\mathrm{EXT}}_{\mathcal{PBS}}$. 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 IND and UF. In [7] 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.

4.1 Generic Constructions

We now show how to construct policy-based signatures satisfying simulatability and extractability (and, by Theorem 1, IND and UF) for any **NP**-relation PC. In [7] we show that the assumptions we make are not only sufficient but necessary.

An first approach could be the following, similar to the generic construction of group signatures in [10]: 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). 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. 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 IND and UF, 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[3] and makes a zero-knowledge proof π that he knows either (I) an issuer

[3] 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 $(crs, pk, mvk) \leftarrow pp$
Parse $(ovk, C_p, C_s, C_w, \pi, \tau) \leftarrow \sigma$
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)

Fig. 3. Generic construction of PBS

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

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

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

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

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

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

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

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

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

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

Fig. 4. PBS based on SE-NIZKs.

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 [7] we prove the following:

Theorem 2. *If $\mathcal{PKE}$ satisfies IND-CPA, $\mathcal{S}ig$ is UF-CMA , $\mathcal{OTS}ig$ is a strongly unforgeable one-time signature scheme and $\mathcal{NIZK}$ is a NIZK proof system for $\mathcal{L}(R_{NP})$ then $\mathcal{PBS}$, 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 [30] (see [7] for the definition). Let $\mathcal{S}ig = (\mathsf{KeyGen}_{\mathrm{sig}}, \mathsf{Sign}_{\mathrm{sig}}, \mathsf{Verify}_{\mathrm{sig}})$ be a signature scheme and for a policy checker PC let $\mathcal{NIZK} = (\mathsf{Setup}_{\mathrm{nizk}}, \mathsf{Prove}, \mathsf{Verify}_{\mathrm{nizk}}, \mathsf{SimSetup}_{\mathrm{nizk}}, \mathsf{SimProve}, \mathsf{Extr}_{\mathrm{nizk}})$ 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

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

Then the scheme in Figure 4 is a PBS for PC which satisfies SIM+EXT. In [7] we prove this for a more general scheme allowing delegation.

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 [31] let us prove that there exists a set of elements $(\boldsymbol{X}, \boldsymbol{Y}) = (X_1, \ldots, X_n, Y_1, \ldots, Y_\ell) \in \mathbb{G}^n \times \mathbb{H}^\ell$ which satisfy equations $\mathsf{E}(\underline{\boldsymbol{X}}, \underline{\boldsymbol{Y}})$ of the form

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

Such an equation E is called a *pairing-product equation*[4] (PPE) and is uniquely defined by its constants $\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{A}, \boldsymbol{B}$ 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 here: they can define the verification predicate of a digital signature (see [1]), or witness the fact that a ciphertext encrypts a certain value (see [7]). 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 $(\boldsymbol{X}, \boldsymbol{Y})$, algorithm $\mathsf{Prove}_{\mathrm{gs}}$ outputs a proof π. Proofs are verified by $\mathsf{Verify}_{\mathrm{gs}}(crs, \mathsf{E}(\underline{\cdot}, \underline{\cdot}), \pi)$. Under the SXDH assumption (see [31]), proofs are *witness-indistinguishable* [27], that is, proofs for an equation using different witnesses are computationally indistinguishable. Moreover, they are *extractable* and thus proofs of knowledge [24]: From every valid proof π, $\mathsf{Extr}_{\mathrm{gs}}(xk, \mathsf{E}(\underline{\cdot}, \underline{\cdot}), \pi)$ extracts a witness $(\boldsymbol{X}, \boldsymbol{Y})$ such that $\mathsf{E}(\boldsymbol{X}, \boldsymbol{Y}) = 1$.

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:

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

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

Our second building block are structure-preserving signatures [1], which were designed to be combined with GS proofs: their keys, messages and signatures consist of elements from $\mathbb{G}$ and $\mathbb{H}$ and signatures are verified by evaluating

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

PPEs. GS proofs let us prove knowledge of keys, messages, and/or signatures which satisfy 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 $\mathcal{S}ig_{\text{sp}}$.

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 [30]. The details of the construction can be found in [7].

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 option would be to additionally demand that an employee hold a credential (verified via PPEs), which she must use as a witness when signing.

5 Applications and Implications

Here we illustrate how PBS 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. Here we show how PBS allows one to easily obtain group signatures [10]. In [7] we show that they imply signatures of knowledge [21] and attribute-based signatures [34]. 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 [7] we show that PBSs imply simulation-extractable NIZKs and IND-CPA encryption. By a result [39] they thus imply IND-CCA public-key encryption. The sufficient assumptions we make in our constructions of Section 4.1 are thus also necessary.

CCA-Secure Group Signatures from PBS. Group signatures [22] 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. As defined in [10], a group-signature scheme $\mathcal{GS}$ is a 4-tuple of PT algorithms. On input 1^λ and the group size 1^n, 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 $m \in \{0,1\}^*$, 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 $\perp$.

Full anonymity requires that an adversary cannot decide which of two group members of its choice produced a group signature, even when given an oracle that opens any other signature. Traceability means that an adversary, which is allowed to corrupt users, cannot produce a group signature which opens to a user that was not corrupted. (We give a formal definition in [7].)

We now construct group signatures from CCA-secure public-key encryption and PBS. Since the former can be constructed from PBS (as we show in [7]), 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 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 $\mathcal{PKE} = (\mathsf{KeyGen}_{\mathrm{pke}}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme satisfying IND-CCA and let $\mathcal{PBS} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathrm{pbs}}, \mathsf{Sign}, \mathsf{Verify})$ be a PBS for the following **NP**-relation:

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

(See [7] for an encryption scheme such that (4) lies in the language of our efficient PBS from Section 4.2.) In [7] we sow that the following group-signature scheme satisfies full anonymity and traceability as formalized by [10].

$\underline{\mathsf{GKg}(1^\lambda, 1^n)}$
$(pp, msk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathrm{pke}}(1^\lambda)$
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 $(gpk \leftarrow (pp, ek), gmsk \leftarrow dk, \boldsymbol{gsk})$

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

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

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

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, 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.

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.

In [7] we provide a syntax and definitions of UF and IND, as well as SIM and EXT, which are straightforward generalizations of those for PBS. However, we strengthen IND by letting the adversary (who obtains msk) construct the keys under one of which the experiment makes a signature. This ensures that when Alice delegates different keys to Bob and Carol, she will not be able to tell by whom a message was signed. Analogously, we let the adversary choose the key in SIM.

With regard to a construction, we note that 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* [33]. 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 can be constructed from any signature scheme. Holding a key, which is a signature on a vector of policies $\boldsymbol{p}$, a user can delegate the key after (possibly) appending a new policy.

Due to space constraints, the definitions as well as the constructions are deferred to the full version [7].

Acknowledgments

Mihir Bellare was supported in part by NSF grants CNS-1228890, CNS-1116800, CNS-0904380 and CCF-0915675. Georg Fuchsbauer was 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.

References

1. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236. Springer, Aug. 2010.
2. S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption: New perspectives and lower bounds. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013*, volume 8043 of *LNCS*, pages 500–518. Springer, Aug. 2013.

3. M. Backes, S. Meiser, and D. Schröder. Delegatable functional signatures. Cryptology ePrint Archive, Report 2013/408, 2013.
4. W. Bagga and R. Molva. Policy-based cryptography and applications. In *Financial Cryptography and Data Security*, pages 72–87. Springer, 2005.
5. M. Barbosa and P. Farshim. On the semantic security of functional encryption schemes. In *PKC 2013*, volume 7778 of *LNCS*, pages 143–161, 2013.
6. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In R. Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Mar. 2008.
7. M. Bellare and G. Fuchsbauer. Policy-based signatures. Cryptology ePrint Archive, Report 2013/413, 2013.
8. M. Bellare and S. Goldwasser. New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 194–211. Springer, Aug. 1989.
9. M. Bellare, S. Meiklejohn, and S. Thomson. Key-versatile signatures and applications: RKA, KDM and Joint Enc/Sig. Cryptology ePrint Archive, Report 2013/326, 2013.
10. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, May 2003.
11. M. Bellare and A. 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.
12. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
13. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, May / June 2006.
14. A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 60–79. Springer, Mar. 2006.
15. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Mar. 2011.
16. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. Cryptology ePrint Archive, Report 2013/352, 2013.
17. X. Boyen. Mesh signatures. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 210–227. Springer, May 2007.
18. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. Cryptology ePrint Archive, Report 2013/401, 2013.
19. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, May 2001.
20. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. Cryptology ePrint Archive, Report 2013/179, 2013.
21. M. Chase and A. Lysyanskaya. On signatures of knowledge. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Aug. 2006.

22. D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Apr. 1991.
23. A. De Caro, V. Iovino, A. Jain, A. O'Neill, O. Paneth, and G. Persiano. On the achievability of simulation-based security for functional encryption. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013*, volume 8043 of *LNCS*, pages 519–535. Springer, Aug. 2013.
24. A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof systems. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 52–72. Springer, Aug. 1987.
25. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
26. Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Efficient public-key cryptography in the presence of key leakage. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 613–631. Springer, Dec. 2010.
27. U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990.
28. G. Fuchsbauer and D. Pointcheval. Anonymous proxy signatures. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 201–217. Springer, Sept. 2008.
29. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
30. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In X. Lai and K. Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Dec. 2006.
31. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Apr. 2008.
32. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. Cryptology ePrint Archive, Report 2013/379, 2013.
33. E. Kiltz, A. Mityagin, S. Panjwani, and B. Raghavan. Append-only signatures. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 434–445. Springer, July 2005.
34. H. K. Maji, M. Prabhakaran, and M. Rosulek. Attribute-based signatures. In A. Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 376–392. Springer, Feb. 2011.
35. M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In *ACM CCS 96*, pages 48–57. ACM Press, Mar. 1996.
36. C. Matt and U. Maurer. A constructive approach to functional encryption. Cryptology ePrint Archive, Report 2013/559, 2013.
37. A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.
38. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Dec. 2001.
39. A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, Oct. 1999.