Warrant-Hiding Delegation-by-Certificate Proxy Signature Schemes[⋆]

Christian Hanser and Daniel Slamanig

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology (TUG), Inffeldgasse 16a, 8010 Graz, Austria
{`christian.hanser|daniel.slamanig@tugraz.at`}

Abstract. Proxy signatures allow an entity (the delegator) to delegate his signing capabilities to other entities (called proxies), who can then produce signatures on behalf of the delegator. Typically, a delegator may not want to give a proxy the power to sign *any* message on his behalf, but only messages from a well defined *message space*. Therefore, the so called *delegation by warrant* approach has been introduced. Here, a *warrant* is included into the delegator's signature (the so called certificate) to describe the message space from which a proxy is allowed to choose messages to produce valid signatures for. Interestingly, in all previously known constructions of proxy signatures following this approach, the warrant is made explicit and, thus, is an input to the verification algorithm of a proxy signature. This means, that a verifier learns the entire message space for which the proxy has been given the signing power. However, it may be desirable to hide the remaining messages in the allowed message space from a verifier. This scenario has never been investigated in context of proxy signatures, but seems to be interesting for practical applications. In this paper, we resolve this issue by introducing so called *warrant-hiding proxy signatures*. We provide a formal security definition of such schemes by augmenting the well established security model for proxy signatures by Boldyreva et al. Furthermore, we discuss strategies how to realize this warrant-hiding property and we also provide two concrete instantiations of such a scheme. They enjoy different advantages, but are both entirely practical. Moreover, we prove them secure with respect to the augmented security model.

Keywords: Proxy signatures, digital signatures, warrant-hiding, zero-knowledge sets, vector commitments, randomized Merkle trees, polynomial commitments

1 Introduction

Proxy signatures, first introduced in [12], allow an entity (the delegator) to delegate his signing capabilities to other entities (called proxies), who can then produce signatures on behalf of the delegator. This concept has seen a considerable amount of interest since its introduction and numerous (secure) constructions have been proposed, see, e.g., [5]. Surprisingly, only quite recently a suitable security model for proxy signatures has been introduced [4], and adopted to multi-level and identity-based proxy signature schemes later on [15]. Apart from standard proxy signatures, various other "flavors" of proxy signatures have been introduced, including threshold [20], one-time [19], blind [17], ring [2], designated-verifier [18] as well as anonymous proxy signatures [9].

In the initial paper [12], it was already observed that the delegator may not want to give a proxy the power to sign *any* message on behalf of the delegator, but only to sign messages from a well defined *message space*. To realize this feature, [12] introduced the so called *delegation by warrant* approach. Here, a signed *warrant* is included into the delegator's signature (the certificate) to describe the delegation. Thereby, any type of security policy may be included into this warrant to enforce the restrictions under which the delegation is valid. This approach seems to be particularly attractive and received the most attention, since

[⋆] This is the full version of a paper to appear at INDOCRYPT'13.

the delegator can clearly define a *message space* for which the signing rights are delegated to the proxy. In state of the art schemes [15,5], a *warrant* ω is either the concatenation of all permitted messages or an abstract description of the message space for which signing is being delegated, together with a certificate, which is a signature on ω (and typically other values including the proxy's identity and public key), under the delegator's private signing key. An abstract description of a message space, thereby, could, for instance, be a context-free grammar, a regular expression, or as in [4], the description of a polynomial-time Turing machine computing the characteristic function of all potential messages, i.e., given a message to decide, whether the message is covered by ω or not.

Problem Statement: This plain inclusion of the warrant into the certificate, however, means that a verifier obtains a precise description of the message space a proxy is allowed to sign. However, this "feature" may not be desirable in some situations. Consider for instance a proxy, who is delegated the rights to sign a contract on behalf of the delegator and the proxy was given the power to sign different versions of the contract, e.g., including different contract conditions such as prices. Given such a signature from the proxy for one of these versions, the warrant would leak all conditions, e.g., more expensive prices, the delegator would have been willing to pay. Consequently, it would be desirable from the point of view of the delegator to *hide* the remaining options from the allowed message space, i.e., the warrant, from a verifier. Otherwise, this could compromise the delegator, as a verifier can learn for instance that the delegator would have been willing to pay a much higher price than he actually did.

In order to overcome this problem, which exists in all known proxy signature schemes supporting the delegation by warrant feature, we introduce the notion of a *warrant-hiding proxy signature scheme*. Basically, in such a signature scheme a proxy learns the warrant, but a proxy signature does not reveal the warrant, as it is not required as an explicit input to the proxy verification algorithm. Consequently, the warrant cannot be determined by a verifier when given a proxy signature. However, a proxy should only be able to produce valid signatures for messages that are *consistent* with the warrant, i.e., messages in the message space defined by the delegator. Thus, there must be an implicit mechanism to check membership in the warrant for a *given* message with corresponding proxy signature, but there should be no means to check membership for other messages.

Contribution: In this paper, we formally define *warrant-hiding proxy signature schemes* by augmenting the state of the art security model of [5] and introducing an additional property for proxy signatures called *privacy*. The latter property captures the fact that given a proxy signature, it is not possible to determine the warrant under which the proxy signer has produced the proxy signature. More precisely, this means that guessing the remaining messages in the warrant is intractable. Basically, this can be achieved by committing to the permitted message space and the proxy needs to provide a non-interactive proof that the message he has signed is contained in the warrant without revealing any other information about the remaining messages in the warrant. Consequently, we can base such a construction generally on zero-knowledge sets [13]. Since general constructions thereof heavily suffer from problems with efficiency, we propose two concrete practical constructions of such a proxy signature scheme based on the delegation-by-certificate approach. Our instantiations can be constructed from any secure digital signature scheme, a randomized version of the Merkle trees yielding hiding vector commitments (without updates) and the secure unconditionally hiding polynomial commitment scheme from [11] respectively. Note that in contrast to a naive approach, i.e., computing an independent certificate (signature of the delegator) for every allowed message, our approach uses a *single* certificate for all messages. After presenting our constructions, we compare them in terms of computational effort as well as bandwidth and prove the security of our constructions in the proposed security model. Finally, we mention open problems for future work.

Outline: Section 2 discusses the cryptographic preliminaries. In Section 3, we present the formal framework for proxy signatures, the security model and our extensions to cover warrant-hiding proxy signature schemes. Section 4 discusses general design strategies and presents our two constructions of warrant-hiding proxy signatures as well as a comparison of their efficiency. Finally, Section 5 concludes the paper and lists open issues for future work.

2 Preliminaries

2.1 Basic Notions

Here, we briefly recall the definitions of bilinear maps, the t-SDH assumption, standard digital signature schemes as well as pseudorandom generators.

Definition 1 (Bilinear Map). Let G, G_T be two cyclic groups of the same prime order p, where G is additive and G_T is multiplicative. We call the map $e : G \times G \to G_T$ a *symmetric bilinear map* or *symmetric pairing* if it is efficiently computable and the following conditions hold:

Bilinearity: For all $P_1, P_2 \in G$ we have for all $P \in G$:
$e(P_1 + P_2, P) = e(P_1, P) \cdot e(P_2, P)$ and $e(P, P_1 + P_2) = e(P, P_1) \cdot e(P, P_2)$.
Non-degeneracy: If P generates G, then $g = e(P, P)$ generates G_T, i.e. $g \neq 1$.

In practice, G and G_T will typically be a suitable elliptic curve group of prime order and a torsion subgroup of some suitable finite field, respectively.

Definition 2 (t-Strong Diffie Hellman Assumption (t-SDH)). Let p be a prime of bitlength κ, G be a p-order group, $\alpha \in_R \mathbb{Z}_p^*$ and let $(P, \alpha P, \ldots, \alpha^t P) \in G^{t+1}$ for some $t > 0$. Then, for every PPT adversary $\mathcal{A}$ there is a negligible function ϵ such that

$$\Pr\left(\mathcal{A}(P, \alpha P, \alpha^2 P, \ldots, \alpha^t P) = \left(c, \frac{1}{\alpha + c} P\right)\right) \leq \epsilon(\kappa)$$

for any $c \in \mathbb{Z}_p \setminus \{-\alpha\}$.

Definition 3 (Digital Signature Scheme). A *digital signature scheme* DSS is a triple $(\mathsf{K}, \mathsf{S}, \mathsf{V})$ of PPT algorithms:

$\mathsf{K}(\kappa)$**:** Is a key generation algorithm that takes input a security parameter $\kappa \in \mathbb{N}$ and outputs a private (signing) key sk and a public (verification) key pk.
$\mathsf{S}(m, \mathsf{sk})$**:** Is a (probabilistic) algorithm, which takes as input a message $M \in \{0,1\}^*$ and a private key sk, and outputs a signature σ.
$\mathsf{V}(\sigma, m, \mathsf{pk})$**:** Is a deterministic algorithm, which takes as input a signature σ, a message $M \in \{0,1\}^*$ and a public key pk, and outputs a single bit $b \in \{\texttt{true}, \texttt{false}\}$ indicating whether σ is a valid signature for M under pk.

Furthermore, we require the digital signature scheme to be correct, i.e., for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{K}(\kappa)$ and all $M \in \{0,1\}^*$ we have $\mathsf{V}(\mathsf{S}(M, \mathsf{sk}), M, \mathsf{pk}) = \texttt{true}$. A digital signature scheme is *secure*, if it is existentially unforgeable under adaptively chosen-message attacks (UF-CMA) [10]. Note that in practice, the sign and verify algorithms will typically use a hash function to map input messages to constant size strings, which is also known as the *hash-then-sign* paradigm.

Definition 4 (Pseudorandom Generator (PRG)). A *pseudorandom generator* $f : \{0,1\}^\kappa \to \{0,1\}^\ell$, with $\ell > \kappa$ being positive integers, is a function that can be computed in polynomial time. The input s_0 to the function is called *seed.* A PRG is called *secure* if its output is computationally indistinguishable from random when given a random seed s_0.

2.2 Commitments

A *commitment scheme* CS as a tuple $(\mathsf{CSetup}, \mathsf{CCommit}, \mathsf{COpen})$ of PPT algorithms:

$\mathsf{CSetup}(\kappa)$**:** Takes a security parameter κ and produces and outputs public parameters cpk which we assume to be implicitly input to the other two algorithms.

$\mathsf{CCommit(M)}$**:** Takes a value $M \in \mathcal{M}$ and outputs a tuple $(\mathcal{C}, O)$ representing the commitment $\mathcal{C}$ to M and the open information O.

$\mathsf{COpen}(\mathcal{C}, \mathsf{O})$**:** Gets $(\mathcal{C}, O)$ and outputs either $M \in \mathcal{M}$ or $\perp$ to indicate success or failure, respectively.

A commitment scheme is required to be *hiding* and *binding*. The former means that the value $M \in \mathcal{M}$ is hidden in $\mathcal{C}$ unless the open information O is available, whereas the latter means that is not possible to find an open information O' such that the given commitment $\mathcal{C}$ opens to $M' \neq M$. Besides hiding and binding, a commitment scheme needs to be *correct*, which means that for every honestly computed commitment $\mathcal{C}$, we have $\mathsf{COpen}(\mathsf{CCommit}(M)) = M$ for all $M \in \mathcal{M}$.

We call a commitment scheme *r-binding*, if it satisfies correctness, hiding and *relaxed-binding* [1]. Relaxed-binding is a weaker notion than binding and uses a modified security game. In this game the adversary A choses a value M. Then, is is given a commitment $(\mathcal{C}, O)$ to M (the randomness r for $\mathsf{CCommit}$ is randomly chosen but not controlled A). If r-binding holds, A is not able to efficiently find O' computed with randomness r' such that $\mathsf{COpen}(\mathcal{C}, O') = M' \neq M$. We write $\mathsf{CCommit}(M, r)$ if we address an r-binding commitment scheme.

In order to support larger messages as input to the $\mathsf{CCommit}$ algorithm, it is common to use the so called *hash-then-commit* approach. It is not hard to see that this approach yields a secure commitment scheme assuming the existence of secure hash functions (collision resistance) and the security of the underlying commitment scheme. Subsequently, whenever we use commitment schemes, we assume that the hash-then-commit paradigm is being implicitly applied.

2.3 Polynomial Commitments

The constant-size unconditionally hiding $\mathsf{PolyCommit}_{\mathsf{Ped}}$ polynomial commitment scheme from [11] is based on Pedersen commitments [14] and uses an algebraic property of polynomials $f(X) \in \mathbb{Z}_p[X]$. Namely, that $(X - \gamma)$ perfectly divides the polynomial $f(X) - f(\gamma)$ for $\gamma \in \mathbb{Z}_p$. We briefly recall the construction:

$\mathsf{PSetup}(\kappa, t)$**:** Pick two groups G, G_T of the same prime order p (with p being a prime of bitlength κ) having a symmetric pairing $e : G \times G \rightarrow G_T$ such that the t-SDH assumption holds. Choose two generators $P, Q \in G$ and $\alpha \in_R \mathbb{Z}_p^*$ and output $\mathsf{ppk} = (G, G_T, p, e, P, \alpha P, \ldots, \alpha^t P, Q, \alpha Q, \ldots, \alpha^t Q)$ as well as $\mathsf{psk} = \alpha$.

$\mathsf{PCommit}(\mathsf{ppk}, f(X))$**:** Given $f(X) \in \mathbb{Z}_p[X]$ with $\deg(f) \leq t$, pick a random polynomial $r(X) \in \mathbb{Z}_p[X]$ with $\deg(f) \leq \deg(r) \leq t$ and compute the commitment $\mathcal{C} = f(\alpha)P + r(\alpha)Q \in G$ and output $\mathcal{C}$.

$\mathsf{POpen}(\mathsf{ppk}, \mathcal{C}, f(X), r(X))$**:** Output $(f(X), r(X))$.

$\mathsf{PVerify}(\mathsf{ppk}, \mathcal{C}, f(X), r(X))$**:** Verify whether

$$\mathcal{C} = \sum_{i=0}^{\deg(f)} f^{(i)}(\alpha^i P) + \sum_{i=0}^{\deg(r)} r^{(i)}(\alpha^i Q)$$

holds and output `true` on success and `false` otherwise.

$\mathsf{PCreateWit}(\mathsf{ppk}, f(X), r(X), \gamma)$**:** Compute $\phi(X) = \frac{f(X)-f(\gamma)}{X-\gamma}$, $\hat{\phi}(X) = \frac{r(X)-r(\gamma)}{X-\gamma}$ and $W_\gamma = \phi(\alpha)P + \hat{\phi}(\alpha)Q$ and output $(\gamma, f(\gamma), r(\gamma), W_\gamma)$.

$\mathsf{PVerifyWit}(\mathsf{ppk}, \mathcal{C}, \gamma, f(\gamma), r(\gamma), W_\gamma)$**:** Verify that $f(\gamma)$ is the evaluation of unknown f at point γ. This is done by checking whether

$$e(\mathcal{C}, P) = e(W_\gamma, \alpha P - \gamma P) \cdot e(f(\gamma)P + r(\gamma)Q, P)$$

holds. Output `true` on success and `false` otherwise.

A polynomial commitment scheme is *secure* if it is correct, polynomial binding, evaluation binding and hiding. This scheme can be proven secure under the t-SDH assumption in G. Note that α must remain unknown to the committer (and thus the setup must be run by a TTP), since, otherwise, it would be a trapdoor commitment scheme.

In one of our constructions, we require an r-binding variant of $\mathsf{PolyCommit}_{\mathsf{Ped}}$, since the random polynomial required for the hiding is not chosen implicitly in $\mathsf{PCommit}$, but provided externally by computing it from a compact seed using a PRG.

Lemma 1. *The aforementioned modification of* $\mathsf{PolyCommit}_{\mathsf{Ped}}$ *satisfies the r-binding property.*

Proof. In order to show that the r-binding property holds for this variant of $\mathsf{PolyCommit}_{\mathsf{Ped}}$, we can follow the same strategy used to prove the binding of $\mathsf{PolyCommit}_{\mathsf{Ped}}$ in [11]. Note, that now the adversary is allowed to choose $f(X)$, but $r(X)$ is randomly chosen by the challenger. Then, $\mathcal{C}, f(X), r(X)$ is given to the adversary and the adversary needs to deliver $f'(X), r'(X)$ with $f'(X) \neq f(X)$ such that $\mathsf{PVerify}(\mathsf{ppk}, \mathcal{C}, f'(X), r'(X))$ returns `true`. It is not hard to see, that the r-binding of this variant of $\mathsf{PolyCommit}_{\mathsf{Ped}}$ can be proven using the the same reduction to the DL problem in G as in [11]. □

2.4 Randomized Merkle Trees

Let T be a complete binary tree of height h with n leaves and let N be the set of nodes of T. Furthermore, let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a secure hash function, $\lambda : N \to \{0,1\}^\ell$ be a labeling function, κ be a security parameter and $(\mathsf{CSetup}, \mathsf{CCommit}, \mathsf{COpen})$ be an unconditionally hiding commitment scheme CS producing commitments of length ℓ. Then, T is called *randomized Merkle tree* if the labeling function λ is recursively defined as follows:

$$\lambda(v) = \begin{cases} H(\lambda(v_L)||\lambda(v_R)) & \text{if } v \text{ has two children } v_L, v_R, \\ H(\lambda(v_L)) & \text{if } v \text{ has one child } v_L, \text{ and} \\ \mathcal{C}_i & \text{if } v \text{ is the } i\text{'th leaf,} \end{cases}$$

where $(\mathcal{C}_i, O_i) = \mathsf{CCommit}(M_i)$ and $\mathcal{M} = (M_i)_{i=1}^n$ is the sequence of strings assigned to the leaves.

Let us additionally define the *authentication path* or *witness* of a leaf i with label $\mathcal{C}_i$ as $W_{M_i} = (w_j)_{j=1}^h$, where the value w_j at height j is defined to be the label of the sibling of the node of height j at the unique path from $\mathcal{C}_i$ to the root.

2.5 Hiding Vector Commitments from Randomized Merkle Trees

Vector commitments allow to commit to an ordered sequence of values represented as a compact commitment and to selectively open values at given positions. For a detailed description of vector commitments, we refer the reader to [6]. Below, we present a novel construction of vector commitments from randomized Merkles trees which are additionally hiding, but do not support updates and proofs of updates. Yet, it seems to be quite straight forward to modify the construction in order to support these two operations by replacing the leaf commitments with trapdoor commitments (chameleon hashes). Our construction uses an r-binding commitment scheme for the leaves, since we do not want the randomizers $\mathcal{R}$ required for the hiding to be generated implicitly, but from a compact seed. Clearly, one could also create a hiding vector commitment scheme in the ordinary sense using a binding and unconditionally hiding commitment scheme for the leaves.

Subsequently, let $\mathsf{VectorCommit}_{\mathsf{Merkle}} = (\mathsf{VKeyGen}, \mathsf{VCommit}, \mathsf{VOpen}, \mathsf{VVerify})$ be a tuple of PPT algorithms associated with a randomized Merkle tree T, such that:

$\mathsf{VKeyGen}(\kappa)$**:** Given the security parameter κ, run $\mathsf{CSetup}(\kappa)$ to obtain cpk of a suitable unconditionally hiding r-binding commitment scheme and output cpk.

$\mathsf{VCommit}(\mathcal{M}, \mathcal{R})$**:** Given a sequence of n messages $\mathcal{M}$ and a sequence of n randomizers $\mathcal{R}$, output the root hash $\mathcal{C}$ of T.

$\mathsf{VOpen}(i, \mathcal{M}, \mathcal{R})$**:** Takes a leaf index i, a sequence of n messages $\mathcal{M}$ and a sequence of n randomizers $\mathcal{R}$ and outputs the authentication path W_{M_i}.

$\mathsf{VVerify}(\mathcal{C}, i, M_i, r_i, W_{M_i})$**:** Takes a root hash $\mathcal{C}$ of a randomized Merkle tree T, the leaf index i, a message M_i, the randomizer r_i and an authentication path W_{M_i} and returns `true` if $\mathcal{C}$ equals the root hash reconstructed from M_i and the authentication path W_{M_i} and `false` otherwise.

We note that the auxiliary information in [6] essentially corresponds to $(\mathcal{M}, \mathcal{R})$ in our case. The security requirements are *correctness* and *position r-binding* as in [6] and additionally *hiding*.

Theorem 1. *Assuming the existence of secure hash functions and of secure, unconditionally hiding r-binding commitment schemes, the above construction of a hiding, r-binding vector commitment scheme is secure.*

Proof. The proof of the scheme's correctness can easily be verified. Since the above construction is essentially what Steinfeld et. al implicitly use in [16], the hiding as well as the r-binding properties follow directly from the proofs in [16]. The position binding of the above construction is immediate due to the structure of the Merkle tree (and the security of the used hash function) and the r-binding of the leaf commitments. □

3 Proxy Signatures

In this section, we recall the formal model for proxy signatures and the security model of [4]. Then, we present an additional definition, capturing the warrant-hiding property and, finally, we define what constitutes a secure *warrant-hiding proxy signature scheme.*

Definition 5 (Proxy Signature Scheme [4]). A *proxy signature scheme* is a tuple $\mathsf{PSS} = (\mathsf{DSS}, (\mathsf{D}, \mathsf{P}), \mathsf{PS}, \mathsf{PV}, \mathsf{ID})$ of PPT algorithms and the algorithms $(\mathsf{D}, \mathsf{P}), \mathsf{PS}, \mathsf{PV}, \mathsf{ID}$ are given access to a potentially empty common reference string P.[1] Furthermore, DSS is a secure digital signature scheme and the other algorithms are defined as follows:

- (D, P) is a pair of interactive probabilistic algorithms forming the (two-party) proxy-designation protocol. Each algorithm gets the two public keys pk_i, pk_j for the delegator i and the proxy j, respectively, as input. D also takes as input the private key sk_i of the delegator, the identity j of the proxy, and a message space descriptor (warrant) ω for which user i wants to delegate its signing rights to user j. P also takes as input the private key sk_j of the proxy. As a result of the interaction, the expected local output of P is skp, a proxy signing key that user j uses to produce proxy signatures on behalf of user i, for messages in ω. D has no local output. We write $\mathsf{skp} = (\mathsf{D}(\mathsf{pk}_i, \mathsf{sk}_i, j, \mathsf{pk}_j, \omega), \mathsf{P}(\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{pk}_i))$ for the result of this interaction.
- PS is the (probabilistic) proxy signing algorithm. As input it takes a proxy signing key skp and a message $M \in \{0,1\}^*$ and outputs a proxy signature σ_p.
- PV is the deterministic proxy verification algorithm. It takes a public key pk, a message $M \in \{0,1\}^*$ and a proxy signature σ_p as input, and outputs `true` or `false`. In the former case, we say that σ_p is a valid proxy signature for $M \in \omega$ relative to pk.

[1] If P is empty this definition exactly matches the definition given in [4,5].

- ID is the proxy identification algorithm. It takes input a valid proxy signature σ_p, and outputs an identity $j \in \mathbb{N}$ or $\perp$ in case of an error.

As it is required by proxy signature schemes when used in practice, we assume the existence of a public key infrastructure. This means that the public keys of delegators and proxies are available in an authentic fashion, i.e., bound to their identities, to all participants.

Definition 6 (Security of a Proxy Signature Scheme [4]). Let $\mathsf{PSS} = (\mathsf{DSS}, (\mathsf{D}, \mathsf{P}), \mathsf{PS}, \mathsf{PV}, \mathsf{ID})$ be a proxy signature scheme, A be an adversary and $\kappa \in \mathbb{N}$. We associate to PSS, A and κ the following game. First, if required, a TTP generates the common reference string P and makes it publicly available (we then implicitly assume that the challenger as well as A have access to P). Then, a public and private key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$ for user 1 is generated via $\mathsf{K}(\kappa)$ and a counter n for the number of users is initialized to 1. The game initializes an empty array skp_1 to store the self-delegated proxy signing keys and corresponding message spaces, and empty sets DU and CS. The set DU stores the identities of the users designated by user 1 (together with the message spaces for which they are designated). The set CS keeps track of the set of messages for which the adversary can produce proxy signatures by user 1 on behalf of user 1 using compromised self-delegated proxy signing keys. Adversary A is given input pk_1 and it can make the following requests or queries in any order and any number of times:

- (i registers pk_i) A can request to register a public key pk_i for user $i = n+1$ by outputting pk_i. The key is stored, counter n is incremented, and an empty array skp_i is created. This array will store the proxy signing keys of user 1 on behalf of user i together with the message spaces ω to which they correspond.
- (1 designates i) A can request to interact with user 1 running algorithm $\mathsf{D}(\mathsf{pk}_1, \mathsf{sk}_1, i, \mathsf{pk}_i, \omega)$, for some $i \in \{2, \ldots, n\}$ and some message space ω (chosen by A). In the interaction, A plays the role of user i running $\mathsf{P}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_1)$. After a successful run, DU is set to $DU \cup \{(i, \omega)\}$.
- (i designates 1) A can request to interact with user 1 running $\mathsf{P}(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}_i)$, for some $i \in \{2, \ldots, n\}$. In the interaction, A plays the role of user i running $\mathsf{D}(\mathsf{pk}_i, \mathsf{sk}_i, 1, \mathsf{pk}_1, \omega)$ for some message space ω selected by A. If skp is the resulting proxy signing key, then the pair (skp, ω) is stored in the last unoccupied position of skp_i. A does not have access to the elements in skp_i.
- (1 designates 1) A can request that user 1 runs the designation protocol with itself for some message space ω. A is given the transcript of the interaction. If skp is the resulting proxy signing key, the pair (skp, ω) is stored in the next available position of skp_1.
- (exposure of the l-th proxy signing key produced during self-delegation) A can request to see $\mathsf{skp}_1[l]$ for some $l \in \mathbb{N}$. If $\mathsf{skp}_1[l]$ contains a proxy signing key and message space pair (skp, ω), then skp is returned to A and CS is set to $CS \cup \omega$. Otherwise, $\perp$ is returned to A.
- (standard signature by 1) A can query oracle $\mathsf{O}_\mathsf{S}(\mathsf{sk}_1, \cdot)$ with a message M and obtain a standard signature for M by user 1, $\sigma = \mathsf{S}(M, \mathsf{sk}_1)$.
- (proxy signature by 1 on behalf of i using the l-th proxy signing key) A can make a query (i, l, M), where $i \in [n], l \in \mathbb{N}$ and $M \in \{0,1\}^*$, to oracle $\mathsf{O}_{\mathsf{PS}}((\mathsf{skp}_u)_{u \in [n]}, \cdot, \cdot, \cdot)$. If $\mathsf{skp}_i[l]$ contains a proxy signing key and message space pair (skp, ω), we say the query is valid and the oracle returns $\mathsf{PS}(\mathsf{skp}, M)$. Otherwise, we say the query is invalid and the oracle returns $\perp$.

Eventually, A outputs a forgery (M, σ) or $(M, \sigma_p, \mathsf{pk})$. The output of the game is as follows:

Forgery of a standard signature: If the forgery is of the form (M, σ), where $\mathsf{V}(\sigma, M, \mathsf{pk}_1) = \mathtt{true}$, and M was not queried to oracle $\mathsf{O}_\mathsf{S}(\mathsf{sk}_1, \cdot)$, then return 1.

Forgery of a proxy signature by user 1 on behalf of user $i \neq 1$: If the forgery is of the form $(M, \sigma_p, \mathsf{pk}_i)$, for some $i \in \{2, \ldots, n\}$, where $\mathsf{PV}(\mathsf{pk}_i, M, \sigma_p) = 1$, $\mathsf{ID}(\sigma_p) = 1$, and no valid query (i, l, M), for $l \in \mathbb{N}$, was made to the oracle $\mathsf{O_{PS}}((\mathsf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, then return 1.

Forgery of a proxy signature by user 1 on behalf of user 1: If the forgery is of the form $(M, \sigma_p, \mathsf{pk}_1)$, where $\mathsf{PV}(\mathsf{pk}_1, M, \sigma_p) = 1$, $\mathsf{ID}(\sigma_p) = 1$, no valid query $(1, l, M)$, for $l \in \mathbb{N}$, was made to $\mathsf{O_{PS}}((\mathsf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, and $M \notin CS$ then return 1.

Forgery of a proxy signature by user $i \neq 1$ on behalf of user 1; user i was not designated by user 1 to sign M: If the forgery is of the form $(M, \sigma_p, \mathsf{pk}_1)$, where $\mathsf{PV}(\mathsf{pk}_1, M, \sigma_p) = 1$ and for each message space ω for which $(\mathsf{ID}(\sigma_p), \omega) \in DU$ it holds that $M \notin \omega$ then return 1.

Otherwise, return 0.

A wins the game, if it returns 1. We say that PSS is a secure proxy signature scheme, if the probability of winning the above game is negligible in the security parameter κ for all polynomial-time adversaries A.

For our privacy definition we have chosen an extractability style game instead of an indistinguishability style game, since it is not possible to find a meaningful notion of indistinguishability due to the fact that the adversary must not know the entire warrant. This requires the warrant to be chosen by the challenger.

Definition 7 (Privacy of a Proxy Signature Scheme). Let the setup be identical to the one in Definition 6. In query phase 1, A is allowed to issue the same types of queries as in the unforgeability game. At some point, A signals the challenger that it is ready to proceed to phase 2 by submitting the tuple (i, c) with $c > 1$. Now, the challenger chooses a warrant ω^*, consisting of c random messages from some message space $\mathbb{M}$ of minimum size $c + 1$, computes the proxy signing key $\mathsf{skp}^* = (\mathsf{D}(\mathsf{pk}_1, \mathsf{sk}_1, i, \mathsf{pk}_i, \omega^*), \mathsf{P}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_1))$ and stores the proxy signing key to a new array skp'_i. Then, in query phase 2, A is allowed to issue queries as in phase 1. Additionally, A is allowed to query proxy signatures for the proxy key in skp'_i for all but one message M^* in the warrant ω^* (whereas on receiving query $l \in \{0, \ldots, c-2\}$ the challenger chooses an unused index l and takes message M_l from ω^*). At some point, A outputs a warrant ω' and wins if $\omega' = \omega^*$.

We say that PSS is *warrant-hiding*, if for all polynomial-time adversaries the probability of winning the above game is negligibly close to $1/|\mathbb{M}'|$, where $\mathbb{M}'$ represents the message space $\mathbb{M}$ minus all queried messages.

Definition 8 (Secure Warrant-Hiding Proxy Signature Scheme). If a secure PSS is warrant-hiding with respect to Definition 7, then we call PSS a *secure warrant-hiding* PSS (WHPSS).

4 Warrant-Hiding Proxy Signature Schemes

In this section we give the problem statement, discuss a generic design strategy and present concrete representation of the warrants which are used in our instantiations and present the two schemes.

Problem Statement: When trying to make the warrant implicit and hidden, one must, on the one hand, enforce that proxy signatures are only valid for messages within the warrant, which requires some suitable representation of the warrant, and, on the other hand, that verifiers cannot test messages against this representation by brute force to determine the remaining messages in the warrant. Furthermore, it is desirable that the representation of the warrant is compact, proxy signatures are compact and the verification of proxy signatures does not require interaction with the delegator, i.e., the verification of warrant-hiding proxy signatures should be non-interactive. Particular issues of interest, from the point of view of a designator, are:

- The verification of a proxy signature should hide the remaining messages in the warrant and
- the verification of a proxy signature should should not reveal (too much) information on the exact size of the warrant.

The first issue is satisfied by both of our constructions and concerning the second issue our constructions reveal an upper bound on the size of the warrant. In Section 4.6, we, however, discuss how the second issue can be achieved, although, reducing the efficiency of such a scheme.

A cryptographic building block that immediately comes to mind when being confronted with this problem statement is a commitment scheme. In particular, one seeks a commitment scheme that is capable of committing to a set of values resulting in a compact commitment and allows to selectively prove membership of a value in the commitment, while at the same time hiding the remaining values in the commitment. Primitives that satisfy both aforementioned properties are zero-knowledge sets [13] and vector commitments [6]. Latter, however, needs to be modified in a way such that it supports hiding. Another primitive, which seems suited at first glance, but actually turns out to be unsuitable, is the concept of a cryptographic accumulator. We briefly discuss these primitives in our context below.

(Nearly) Zero-Knowledge Sets: Zero-knowledge sets (ZKS) were introduced by Micali et. al [13]. They allow a prover to commit to an arbitrary finite set S in such a way that for any string x he can provide an efficient proof of whether $x \in S$ or $x \notin S$, without revealing any knowledge beyond this (non) membership. In particular, the verifier of the proof neither learns the remaining elements of the set S and nor the size of S. Follow up work [8,7] has instantiated ZKS from a variety of other assumptions and improved the efficiency. In [11], it is shown that when relaxing ZKS to nearly ZKS, which no longer require hiding an upper bound on the cardinality of S, the size of the proof that an element is (or is not) in a committed set is reduced by a factor of sixteen or more, when compared to the best known ZKS construction. One of our instantiations uses the polynomial commitment scheme introduced in [11]. Furthermore, note that we no not require non-membership proofs in our application and, thus, do not rely on costly general ZKS constructions. Doing so, we obtain a size of the public parameters of $\mathcal{O}(|\omega|)$ and a size of the proxy signature of $\mathcal{O}(1)$.

Hiding Vector Commitments: Vector commitments [6] allow to commit to an ordered sequence of values represented as a compact commitment and to selectively open values at given positions. However, the constructions of [6] do not provide hiding as it is, but can be extended to support hiding by applying the vector commitments to a sequence of hiding commitments instead of messages. They additionally support updates, which is not required in our application. Most importantly, the constructions of [6] require public parameters of size $\mathcal{O}(|\omega|^2)$. Our proposed construction uses an efficient hiding vector commitment from randomized Merkle trees as they are used implicitly in the construction of content extraction signatures in [16]. Our construction does not provide updates as it is. Nevertheless, it seems that it can easily be turned into a vector commitment scheme supporting updates. This could be achieved by exchanging the commitments of the leaves with trapdoor commitments (chameleon hash values). In contrast to the vector commitments of [6], the public parameters are $\mathcal{O}(1)$, but the size of the proxy signatures is $\mathcal{O}(\log |\omega|)$. As it is also the case with our first construction, an upper bound of the size of the warrant is revealed (vector commitments of [6] reveal the exact size).

Cryptographic Accumulators: A cryptographic accumulator [3] allows to represent a finite set of values S by a single value (the accumulator), whose size is independent of the size of S. For every accumulated value, one can compute a witness. Having such a witness, anybody can verify that the corresponding value has indeed been accumulated. However, it is infeasible to find a witness for a value that was not accumulated. The basic problem with accumulators is that they do not guarantee the hiding of the accumulated set S, which is crucial for our application.

4.1 Warrant Representation

In our constructions, the warrant ω is a sequence of messages $\omega = (M_i)_{i=1}^{c}$, which is being mapped into a compact representation, i.e., of constant size, which is then integrated into the certificate of the proxy. We stress that we do not require an explicit ordering and could also use a set representation instead, but we use the sequence notation for a consistent description of both schemes (in the second scheme, the messages are ordered, but the ordering is arbitrary and does not have any meaning for our construction). Note that in contrast to an abstract description of the message space, which allows the representation of a potentially unbounded message space, our construction supports only fixed message spaces in the sense that each message in this space must be known a priori. In particular, the number of messages is polynomially bounded and each message needs to be generated by a polynomial time algorithm. However, in most practical applications of proxy signatures such a message space is sufficient. Furthermore, this allows us to construct proxy signatures, which provide the warrant-hiding property. Considering potentially unbounded message spaces while hiding the warrant seems to be far from trivial and is an interesting aspect for future work.

Polynomial Commitments: Our first construction is based on the constant-size unconditionally hiding polynomial commitment scheme of Kate et al. [11]. Loosely speaking, this construction works in the following way. A delegator maps all c messages in the warrant to a polynomial of degree c, whereas the roots of this polynomial are defined to be hash values of messages in the warrant. Then, the delegator commits to this polynomial and signs the polynomial commitment resulting in a certificate for the proxy. A proxy signer is then allowed to produce proxy signatures by generating witnesses for the roots of the polynomial (representing valid messages in the warrant) and signs the witness along with the public key of the delegator. Then, any verifier can check both signatures and verify whether the message and the witness correspond to the committed polynomial and represent a root of the polynomial. Consequently, the verifier can check the correctness of a proxy signature without learning the remaining messages in the warrant. Latter is due to the unconditional hiding property of the polynomial commitment scheme.

Now, we need to discuss in detail how a set of valid messages (the warrant) is represented. Instead of using $\omega = (M_i)_{i=1}^{c}$ itself, we firstly construct a set $\omega_H = \{H(M_i) : i = 1, \ldots, c\}$, where $H : \{0,1\}^* \to \mathbb{Z}_p$ is a secure hash function. From this set ω_H, we secondly derive the so-called *warrant polynomial* $m(X)$ using the map

$$\phi : 2^{\mathbb{Z}_p} \to \mathbb{Z}_p[X] \quad \text{with} \quad \omega_H \mapsto \prod_{H \in \omega_H} (X - H).$$

Note that the degree of $m(X)$ is polynomially bounded, i.e., represents the size of the warrant.

The intuition for this particular representation is that if the hash values of messages in ω are roots of the warrant polynomial $m(X)$, this polynomial uniquely captures the messages given by the warrant. More precisely, a message is in the warrant if and only if its hash value is a root of $m(X)$, i.e., there is a 1-to-1 correspondence between the set of valid witnesses and the warrant (up to collisions in the hash function H). Otherwise, if the valid messages did not correspond to the roots of $m(X)$ and were arbitrary evaluations, a dishonest proxy signer would be able to generate witnesses for arbitrary messages and, in further consequence, efficiently produce valid proxy signatures for messages outside the warrant.

Hiding Vector Commitments from Randomized Merkle Trees: Our second construction is based on hiding vector commitments from randomized Merkle trees. Loosely speaking, this construction works in the following way. A delegator generates an r-binding commitment for each of the c messages in the warrant and then computes the root of the randomized Merkle tree T with c leaves. This means that T aggregates commitments to all c messages in ω into a single root hash value. Then, the delegator signs the root hash of T resulting in a certificate for the proxy. A proxy signer is then allowed to produce proxy signatures by generating witnesses W_{M_i}, which is the respective authentication path for the leaf $\mathcal{C}_i$, and signs the witness along with the public key of the delegator. Then, any verifier can check both signatures and verify whether the message and the witness correspond to the root hash of T. Consequently, the verifier can check the correctness of a proxy signature without learning the remaining messages in the warrant. Latter is due to the unconditional hiding property of the commitment scheme used at the leaf nodes.

4.2 The First Scheme ($\mathsf{WHPSS}_{\mathsf{PolyCommit}}$)

Before Scheme 1 can be used, a TTP runs a setup in the following way to produce the common reference string, which is then accessible to all parties, i.e., delegators, proxy signers and verifiers, in an authentic fashion:

Setup: Given a security parameter κ and an upper bound $t \in \mathbb{N}$ for the size of the warrant, execute $\mathsf{PSetup}(\kappa, t)$ and obtain $(\mathsf{ppk}, \mathsf{psk})$. Choose a secure PRG $f : \mathbb{Z}_p \to \mathbb{Z}_p^{t+1}$ and output a suitable encoding of the tuple (f, ppk) as the common reference string P.

(D, P): D and P are given local inputs $\mathsf{pk}_i, \mathsf{sk}_i, j, \mathsf{pk}_j, \omega$ and $\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{pk}_i$, where $\omega = (M_i)_{i=1}^{c}$ with $c \leq t$.

D picks a seed $s \in_R \mathbb{Z}_p$, computes $\omega_H = \{H(M_i) : i = 1, \ldots, c\}$ and $m(X) = \phi(\omega_H)$. Then, compute $r(X) \in \mathbb{Z}_p[X]$ with $\deg(r) = \deg(m) = c$ with coefficients obtained evaluating $f(s)$ as well as

$$\mathcal{C} = \mathsf{PCommit}(\mathsf{ppk}, m(X), r(X)) \quad \text{and} \quad \mathsf{cert} = \mathsf{S}(\mathcal{C} \| j \| \mathsf{pk}_j, \mathsf{sk}_i).$$

It sets the proxy signing key of user j as $\mathsf{skp}' = (\mathsf{pk}_i, s, \mathcal{C}, j, \mathsf{pk}_j, \omega, \mathsf{cert})$ and sends it to P. Now, P computes ω_H and $m(X) = \phi(\omega_H)$ as well as $r(X)$ from seed s. It checks whether

$$\mathsf{V}(\mathsf{cert}, \mathsf{PCommit}(\mathsf{ppk}, m(X), r(X)) \| j \| \mathsf{pk}_j, \mathsf{pk}_i) = \mathtt{true}$$

If not, return $\perp$ and terminate. Otherwise, set $\mathsf{skp} = (\mathsf{sk}_j, \mathsf{skp}')$, output skp and terminate. If P returns $\perp$, D aborts. Otherwise, also D terminates correctly.

PS: Given skp, M so that there is an index l with $1 \leq l \leq c$ and $M_l = M$, this algorithm computes ω_H and $m(X) = \phi(\omega_H)$ as well as $r(X)$ from seed s. Then, it computes $h_M = H(M)$, $r_M = r(h_M)$ as well as

$$W_M = \mathsf{PCreateWit}(\mathsf{ppk}, m(X), r(X), h_M) \quad \text{and} \quad \sigma = \mathsf{S}(W_M \| r_M \| \mathsf{pk}_i, \mathsf{sk}_j),$$

and returns $\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j, \mathsf{cert}, \sigma)$.

PV: Given $\mathsf{pk}_i, M, \sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j, \mathsf{cert}, \sigma)$, this algorithm verifies whether

$$\mathsf{V}(\mathsf{cert}, \mathcal{C} \| j \| \mathsf{pk}_j, \mathsf{pk}_i) \;\wedge\; \mathsf{V}(\sigma, W_M \| r_M \| \mathsf{pk}_i, \mathsf{pk}_j) \;\wedge\; \mathsf{PVerifyWit}(\mathsf{ppk}, \mathcal{C}, H(M), 0, r_M, W_M)$$

yields `true`. On success return `true` and `false` otherwise.

ID: Given $\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j, \mathsf{cert}, \sigma)$ output j.

Scheme 1: Warrant-hiding Proxy Signature Scheme from $\mathsf{PolyCommit}_{\mathsf{Ped}}$ ($\mathsf{WHPSS}_{\mathsf{PolyCommit}}$)

4.3 The Second Scheme ($\mathsf{WHPSS}_{\mathsf{VectorCommit}}$)

In contrast to Scheme 1, here the requirement of a TTP for generating a common reference string depends on the commitment scheme used for labeling the leaves of the randomized Merkle tree.

(D, P): D and P are given local inputs $\mathsf{pk}_i, \mathsf{sk}_i, j, \mathsf{pk}_j, \omega$ and $\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{pk}_i$, where $\omega = (M_i)_{i=1}^c$. D picks a seed $s \in_R \{0,1\}^\kappa$, chooses a secure PRG $f : \{0,1\}^\kappa \to (\{0,1\}^\kappa)^c$ and computes $\mathcal{R} = f(s)$ and

$$\mathcal{C} = \mathsf{VCommit}(\omega, \mathcal{R}) \quad \text{and} \quad \mathsf{cert} = \mathsf{S}(\mathcal{C}\|j\|\mathsf{pk}_j, \mathsf{sk}_i).$$

It sets the proxy signing key of user j as $\mathsf{skp}' = (\mathsf{pk}_i, s, \mathcal{C}, j, \mathsf{pk}_j, \omega, \mathsf{cert})$ and sends it to P. Now, P computes $\mathcal{R}$ from s and checks whether

$$\mathsf{V}(\mathsf{cert}, \mathsf{VCommit}(\omega, \mathcal{R})\|j\|\mathsf{pk}_j, \mathsf{pk}_i) = \mathtt{true}$$

If not, return $\perp$ and terminate. Otherwise, set $\mathsf{skp} = (\mathsf{sk}_j, \mathsf{skp}')$, output skp and terminate. If P returns $\perp$, D aborts. Otherwise, also D terminates correctly.

PS: Given skp, M so that there is an index l with $1 \le l \le c$ and $M_l = M$, this algorithm computes $\mathcal{R}$ from seed s, sets $r_M = (r_l, l)$ and computes

$$W_M = \mathsf{VOpen}(l, \omega, \mathcal{R}) \quad \text{and} \quad \sigma = \mathsf{S}(W_M\|r_M\|\mathsf{pk}_i, \mathsf{sk}_j),$$

and returns $\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j, \mathsf{cert}, \sigma)$.

PV: Given $\mathsf{pk}_i, M, \sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j, \mathsf{cert}, \sigma)$ with $r_M = (r_l, l)$, this algorithm verifies whether

$$\mathsf{V}(\mathsf{cert}, \mathcal{C}\|j\|\mathsf{pk}_j, \mathsf{pk}_i) \quad \wedge \quad \mathsf{V}(\sigma, W_M\|r_M\|\mathsf{pk}_i, \mathsf{pk}_j) \quad \wedge \quad \mathsf{VVerify}(\mathcal{C}, l, M, r_l, W_M)$$

yields `true`. On success return `true` and `false` otherwise.

ID: Given $\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j, \mathsf{cert}, \sigma)$ output j.

Scheme 2: Warrant-hiding Proxy Signature Scheme from Vector Commitments ($\mathsf{WHPSS}_{\mathsf{VectorCommit}}$)

4.4 Security

Here, we discuss the security properties of our proposed WHPSS constructions. We are not dealing with the correctness of Scheme 1 and Scheme 2, since this is straight-forward to verify. Subsequently, we informally discuss the security of both constructions and then provide theorems, which we prove in the appendix.

In Scheme 1, the delegator, by running the delegation, commits to a message polynomial $m(X)$ based on an unconditionally hiding polynomial commitment $\mathcal{C}$ using a random polynomial $r(X)$. Hence, since the delegator does not sign the warrant itself, but a representation thereof (the commitment), we need to guarantee that the delegator is not able to change the warrant later on, i.e., finding polynomials $m'(X), r'(X)$ with $m(\alpha) = m'(\alpha)$ as well as $r(\alpha) = r'(\alpha)$, which would violate the binding of $\mathsf{PolyCommit}_{\mathsf{Ped}}$. Now, we argue why the warrant-hiding property holds. Let $\mathcal{C}$ be a commitment to some warrant polynomial $m(X)$ of degree c. Note that $\mathsf{PolyCommit}_{\mathsf{Ped}}$ unconditionally hides $m(X)$ in $\mathcal{C}$ as long as $r(X)$ is unknown ($r(X)$ is only known to the proxy). Along with a proxy signature, a root of $m(X)$ and an evaluation of the random polynomial $r(X)$ are being disclosed. It is not possible to interpolate $r(X)$ unless $c + 1$ (with c being the size of the warrant) distinct evaluations of $r(X)$ are known, which will never happen and, thus, the hiding of $\mathsf{PolyCommit}_{\mathsf{Ped}}$ and the security of the used PRG holds. The warrant polynomial $m(X)$ can only be reconstructed from all c roots, however, then, we no longer need the warrant to be hidden, since then all messages from the warrant are already known. Latter means that the unknown randomizers can not be determined. More formally, in Appendix A we show:

Theorem 2. *Assuming the r-binding of* $\mathsf{PolyCommit}_{\mathsf{Ped}}$*, the existence of secure hash functions and the security of the* DSS *scheme, Scheme 1 is a secure* PSS*.*

Theorem 3. *Assuming the unconditional hiding of* $\mathsf{PolyCommit}_{\mathsf{Ped}}$ *and the existence of secure PRGs, Scheme 1 is a warrant-hiding* PSS*.*

In Scheme 2, the delegator, by running the delegation, commits to a sequence of messages and randomness based on $\mathsf{VectorCommit}_{\mathsf{Merkle}}$ producing a root hash $\mathcal{C}$. As above, due to the binding of the commitment scheme we guarantee that the delegator cannot change the warrant afterwards. The warrant-hiding property holds, because even if an adversary gets to know all the leaf commitments, the respective messages are hidden due to the unconditionally hiding leaf commitment and the security of the used PRG. Latter means that the unknown randomizers can not be determined. More formally, in Appendix A we show:

Theorem 4. *Assuming the r-binding of* $\mathsf{VectorCommit}_{\mathsf{Merkle}}$*, the existence of secure hash functions and the security of the* DSS *scheme, Scheme 2 is a secure* PSS*.*

Theorem 5. *Assuming the unconditional hiding of* $\mathsf{VectorCommit}_{\mathsf{Merkle}}$ *and the existence of secure PRGs, Scheme 2 is a warrant-hiding* PSS*.*

Taking the above results together, we obtain the following result:

Corollary 1. *Scheme 1 and Scheme 2 are both secure* WHPSS*.*

4.5 Efficiency Comparison

In Table 1, we analyze the complexity of both introduced schemes in terms of computational costs of all involved algorithms as well as the sizes of parameters, certificates (delegations) and proxy signatures.

	Computation					**Size**		
Scheme	$\mathcal{D}$	$\mathcal{P}$	$\mathcal{PS}$	$\mathcal{PV}$	$\mathcal{ID}$	P	cert	σ_p
$\mathsf{WHPSS}_{\mathsf{PolyCommit}}$	$\mathcal{O}(\lvert\omega\rvert)$	$\mathcal{O}(\lvert\omega\rvert)$	$\mathcal{O}(\lvert\omega\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\omega\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\mathsf{WHPSS}_{\mathsf{VectorCommit}}$	$\mathcal{O}(\lvert\omega\rvert)$	$\mathcal{O}(\lvert\omega\rvert)$	$\mathcal{O}(\log\lvert\omega\rvert)$	$\mathcal{O}(\log\lvert\omega\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log\lvert\omega\rvert)$

Table 1. Comparison of Costs of Scheme 1 and Scheme 2

We now briefly highlight the major differences between both schemes. In terms of computational effort, $\mathsf{WHPSS}_{\mathsf{PolyCommit}}$ has higher costs for signature generation, but requires constant time for signature verification. In contrast, $\mathsf{WHPSS}_{\mathsf{VectorCommit}}$ is faster in signature generation, but has higher cost for verification (although in practice, the operations are cheap hash function evaluations). In terms of size, $\mathsf{WHPSS}_{\mathsf{PolyCommit}}$ has constant size proxy signatures, but the parameters P generated by a TTP are linear in the warrant size. In contrast, $\mathsf{WHPSS}_{\mathsf{VectorCommit}}$ has small constant size parameters P, but a proxy signature size logarithmic in the size of the warrant. We note that depending on the actual unconditionally hiding commitment scheme used in the construction of the randomized Merkle tree, also here a TTP may be required to be involved in the generation of P, e.g., when using Pedersen commitments.

4.6 On Hiding the Warrant Size

As already noted, both constructions reveal an upper bound on the warrant size. More precisely, in case of $\mathsf{WHPSS}_{\mathsf{PolyCommit}}$ this upper bound is t, which may be adjusted to be larger than any value that allows to draw meaningful conclusions for practical applications. Similarly to above, in case of $\mathsf{WHPSS}_{\mathsf{VectorCommit}}$ one can artificially enlarge the height of the hash tree and introduce dummy leaves to hide the warrant size. Clearly, both cases reduce practicality with increasing the upper bound. In theory, ZKS achieve hiding the cardinality of the set. However, the parameters therefore need to be chosen in a way that they are larger than any meaningful set size, which in practice does not improve on our modifications.

5 Conclusion

In this paper, we have introduced a new type of proxy signatures following the *delegation by warrant* approach. These so called *warrant-hiding proxy signatures* enable a delegator to restrict the message space for a proxy while hiding this message space (warrant) from verifiers.

An interesting question for future work is to construct such signature schemes for potentially unbounded message spaces which do not require exponential effort in producing a delegation. Nevertheless, this does not seem to be straight-forward when seeking efficient constructions suitable for practical applications. It may be interesting to study the security in context of multi-level proxy signatures [15].

Acknowledgements: Both authors have been supported by the European Commission through project FP7-FutureID, grant agreement number 318424.

References

1. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: EUROCRYPT. pp. 83–107 (2002)
2. Awasthi, A.K., Lal, S.: ID-based Ring Signature and Proxy Ring Signature Schemes from Bilinear Pairings. I. J. Network Security 4(2), 187–192 (2007)
3. Benaloh, J.C., de Mare, M.: One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract). In: Advances in Cryptology - EUROCRYPT '93. LNCS, vol. 765, pp. 274–285. Springer (1993)
4. Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. IACR Cryptology ePrint Archive 2003, 96 (2003)
5. Boldyreva, A., Palacio, A., Warinschi, B.: Secure Proxy Signature Schemes for Delegation of Signing Rights. J. Cryptology 25(1), 57–115 (2012)
6. Catalano, D., Fiore, D.: Vector Commitments and Their Applications. In: Public Key Cryptography. LNCS, vol. 7778, pp. 55–72. Springer (2013)
7. Catalano, D., Fiore, D., Messina, M.: Zero-Knowledge Sets with Short Proofs. In: Advances in Cryptology - EUROCRYPT 2008. LNCS, vol. 4965, pp. 433–450. Springer (2008)
8. Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial Commitments with Applications to Zero-Knowledge Sets. In: Advances in Cryptology - EUROCRYPT 2005. LNCS, vol. 3494, pp. 422–439. Springer (2005)
9. Fuchsbauer, G., Pointcheval, D.: Anonymous Proxy Signatures. In: 6th International Conference on Security and Cryptography for Networks, SCN'08. LNCS, vol. 5229, pp. 201–217. Springer (2008)
10. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM J. Comput. 17(2), 281–308 (1988)
11. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Advances in Cryptology - ASIACRYPT 2010. pp. 177–194 (2010)
12. Mambo, M., Usuda, K., Okamoto, E.: Proxy Signatures for Delegating Signing Operation. In: ACM Conference on Computer and Communications Security (CCS 1996). pp. 48–57 (1996)
13. Micali, S., Rabin, M.O., Kilian, J.: Zero-Knowledge Sets. In: Symposium on Foundations of Computer Science (FOCS). pp. 80–91 (2003)
14. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO. pp. 129–140 (1991)
15. Schuldt, J.C.N., Matsuura, K., Paterson, K.G.: Proxy Signatures Secure Against Proxy Key Exposure. In: 11th International Workshop on Practice and Theory in Public-Key (PKC 2008). LNCS, vol. 4939, pp. 141–161 (2008)
16. Steinfeld, R., Bull, L., Zheng, Y.: Content Extraction Signatures. In: Information Security and Cryptology (ICISC 2001). LNCS, vol. 2288, pp. 285–304. Springer (2001)
17. Tan, Z., Liu, Z., Tang, C.: Digital Proxy Blind Signature Schemes Based on DLP and ECDLP. Tech. rep., MM Research Preprints, MMRC, AMSS, Academia, Sinica, Beijing (2002)
18. Wang, G.: Designated-verifier proxy signatures for e-commerce. In: IEEE International Conference on Multimedia and Expo, ICME'04. pp. 1731–1734 (2004)

19. Wang, H., Pieprzyk, J.: Efficient One-Time Proxy Signatures. In: Advances in Cryptology - ASIACRYPT 2003. LNCS, vol. 2894, pp. 507–522. Springer (2003)
20. Zhang, K.: Threshold Proxy Signature Schemes. In: First International Workshop on Information Security, ISW'97. LNCS, vol. 1396, pp. 282–290. Springer (1997)

A Security Proofs of the Constructions

In this section, we provide the proofs of the theorems in Section 4.4. Since from an abstract point of view the algorithms and parameters of Scheme 1 and Scheme 2 are identical, we put the the proofs of Theorem 2 and Theorem 4 as well as Theorem 3 and Theorem 5 into a single proof each and only make case distinctions where necessary (within the corresponding forgery events). For a simpler representation, we use WHPSS to denote either $\mathsf{WHPSS}_{\mathsf{PolyCommit}}$ or $\mathsf{WHPSS}_{\mathsf{VectorCommit}}$ as well as $\mathsf{WCommit}$ to denote the warrant commitment scheme, i.e., either $\mathsf{PolyCommit}_{\mathsf{Ped}}$ or $\mathsf{VectorCommit}_{\mathsf{Merkle}}$.

A.1 Proof of Theorems 2 and 4

The strategy for this proof closely follows the proof of Theorem 4.2 in [5] for the generic construction of delegation-by-certificate proxy signature schemes. For large parts, this proof is analogous to the aforementioned one, but we need to additionally take into account that we do not sign messages but commitments and witnesses for messages in case of certificates and proxy signatures, respectively. However, we augment the entries in the set DU in such a way that they additionally contain the commitments $\mathcal{C}$. For our game, we define the following events analogously to [5]:

E1: A outputs a forgery of the form (M, σ), where $\mathsf{V}(\sigma, M, \mathsf{pk}_1) = \mathtt{true}$, and M was not queried to oracle $O_S(\mathsf{sk}_1, \cdot)$.

E2: A outputs a forgery of the form $(M, \sigma_p, \mathsf{pk}_i)$, where $i \neq 1$, $\sigma_p = (1, \mathcal{C}, W_M, r_M, \mathsf{pk}_1, \mathsf{cert}, \sigma)$, $\mathsf{PV}(\mathsf{pk}_i, M, \sigma_p) = \mathtt{true}$, and no valid query (i, l, M) was made to oracle $O_{\mathsf{PS}}((\mathsf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, for $l \in \mathbb{N}$.

E3: A outputs a forgery of the form $(M, (j, \mathcal{C}, W_M, r_M, \mathsf{pk}', \mathsf{cert}, \sigma), \mathsf{pk}_1)$ such that for all $(j, \omega, \mathcal{C}) \in DU$ with $j \neq 1$ it is the case that $M \notin \omega$.

E4: A outputs a forgery of the form $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}', \mathsf{cert}, \sigma), \mathsf{pk}_1)$ such that pk' was not generated by user 1 during one of the executions of the self delegation protocol.

E5: A outputs a forgery of the form $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}', \mathsf{cert}, \sigma), \mathsf{pk}_1)$ such that pk' was generated by user 1 during one of the executions of the self delegation protocol.

In order to deal with the events **E1, E2, E3**, we introduce adversary B, for event **E4** we introduce adversary C and, finally, for event **E5** we use adversary D. Our goal is to show that $\mathsf{Pr}[\mathbf{E1} \vee \mathbf{E2} \vee \mathbf{E3}]$ is less or equal to the advantage of adversary B, $\mathsf{Pr}[\mathbf{E4}]$ is less or equal to the advantage of adversary C and $\mathsf{Pr}[\mathbf{E5}]$ is less or equal to q_s times the advantage of adversary D. Furthermore, since these events partition the event that adversary A wins our game, the advantage of adversary A is less or equal to the sum of the probabilities of all these events. Consequently, if adversary A produces valid forgeries with non-negligible probability, i.e., one of the events happens, we obtain our desired contradiction to the security of DSS, the r-binding of the respective commitment scheme or the security of H.

Adversary B: Let A be an efficient adversary against WHPSS. We construct an adversary B against DSS, $\mathsf{WCommit}$, or a hash function H. The adversary has access to A's signing oracle under some signing key sk_1, and gets as input the corresponding verification key pk_1. Adversary B simulates the challenger in the game of Definition 6 interacting with adversary A, where the keys of user 1 are $(\mathsf{sk}_1, \mathsf{pk}_1)$. Adversary B works as follows: First, B initializes a counter $n = 1$ that keeps track of the number of users, creates an empty array skp_1, and initializes the sets DU and CS. Note that A and B have access to the common reference string P, if necessary. It then runs A on input pk_1, handling A's requests and answering A's queries as follows:

1. If A requests to register a new user $i = n+1$ by outputting pk_i, then B stores this key, increments n and creates an empty array skp_i.
2. If A requests to interact with $\mathsf{D}(\mathsf{pk}_1, \mathsf{sk}_1, i, \mathsf{pk}_i, \omega)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary message space chosen by A, who plays the role of $\mathsf{P}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_1)$, then B makes a query $\mathcal{C}\|i\|\mathsf{pk}_i$ to its signing oracle $\mathsf{O}_\mathsf{S}(\mathsf{sk}_1, \cdot)$ and receives an answer cert. It forwards $(\omega, \mathcal{C}, s, \mathsf{cert})$ to A and sets DU to $DU \cup \{(i, \omega, \mathcal{C})\}$.
3. If A requests to interact with $\mathsf{P}(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}_i)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary message space chosen by A, who plays the role of $\mathsf{D}(\mathsf{pk}_i, \mathsf{sk}_i, 1, \mathsf{pk}_1)$, then B proceeds as follows. It expects to receive from A a message of the form $(\omega, \mathcal{C}, s, \mathsf{cert})$, B verifies that cert is a valid signature for message $\mathcal{C}\|1\|\mathsf{pk}_1$. If so, B stores $(\omega, \mathcal{C}, s, \mathsf{cert})$ in the last unoccupied position of skp_i. (Notice that, unlike in the real game, adversary B cannot store in skp_i the actual proxy signing keys of user 1 since these keys contain sk_1. However, it is sufficient to store $(\omega, \mathcal{C}, s, \mathsf{cert})$, since this information together with access to the signing oracle under sk_1, is sufficient for producing proxy signatures).
4. If A requests user 1 to run the designation protocol with itself for message space ω, B runs K to obtain $(\mathsf{pk}_1', \mathsf{sk}_1')$ and obtains a signature cert on $\mathcal{C}\|1\|\mathsf{pk}_1'$ from the signing oracle $\mathsf{O}_\mathsf{S}(\mathsf{sk}_1, \cdot)$. It then forwards $(\omega, \mathcal{C}, s, \mathsf{cert})$ to A and stores $(\mathsf{sk}_1', \mathsf{pk}_1, s, \mathcal{C}, 1, \mathsf{pk}_1', \omega, \mathsf{cert})$ in the last unoccupied position of skp_1.
5. If A requests to see $\mathsf{skp}_1[i]$ for some i, then let $(\mathsf{sk}_1', \mathsf{pk}_1, s, \mathcal{C}, 1, \mathsf{pk}_1', \omega, \mathsf{cert})$ be the i'th entry in skp_1. Adversary B forwards $(\mathsf{sk}_1', \mathsf{pk}_1, s, \mathcal{C}, 1, \mathsf{pk}_1', \omega, \mathsf{cert})$ to A and sets CS to $CS \cup \omega$.
6. If A queries its oracle $\mathsf{O}_{\mathsf{S}_1}(\mathsf{sk}_1, \cdot)$ with a message M, B makes query M to its own signing oracle $\mathsf{O}_\mathsf{S}(\mathsf{sk}_1, \cdot)$ and forwards the response to A.
7. If A makes a query (i, l, M), where $i \in [n], l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its proxy signing oracle $\mathsf{O}_\mathsf{PS}((\mathsf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, B responds as follows. If $i = 1$ and $\mathsf{skp}_1[l]$ is not defined then return $\perp$ to A. Otherwise, compute and return to A the quantity $\mathsf{PS}(\mathsf{skp}_1[l], M)$. If $i \neq 1$ and $\mathsf{skp}_i[l]$ is not defined, it returns $\perp$ to A. Otherwise, let $(\omega, \mathcal{C}, s, \mathsf{cert})$ be the content of this position. B submits $(W_M\|r_M\|\mathsf{pk}_i)$ to the signing oracle and obtains in return σ. The proxy signature that B returns to A is $\sigma_p = (1, \mathcal{C}, W_i, r_i, \mathsf{pk}_1, \mathsf{cert}, \sigma)$.

Eventually, A outputs an attempted forgery. Adversary B computes its attempted forgery as follows:

1. If A outputs a forgery of the form (M, σ), then the forgery output by B is (M, σ).
2. If A outputs $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}_1, \mathsf{cert}, \sigma), \mathsf{pk}_i)$ and $i \neq 1$, then adversary B outputs $(\mathsf{pk}_i, \mathsf{pk}_1, \mathcal{C}\|1\|\mathsf{pk}_1, W_M\|r_M\|\mathsf{pk}_i, M, \sigma)$.
3. If A outputs a forgery of the form $(M, \sigma_p, \mathsf{pk}_1)$ with $\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j,\ \mathsf{cert}, \sigma)$ such that $\mathsf{ID}(\sigma_p) = j$ and j is such that for all ω with $(j, \omega, \mathcal{C}) \in DU$, $M \notin \omega$ then B outputs $(\mathsf{pk}_1, \mathsf{pk}_j, \mathcal{C}\|1\|\mathsf{pk}_j, W_M\|r_M\|\mathsf{pk}_1, M, \mathsf{cert}, \sigma)$.

Subsequently, we analyze adversary B. The view of A in this simulation is identical to that in the real game and if one of the given events occurs, then B has won the game.

E1: Clearly, if A is able to produce such a forgery, then it can not have queried the signing oracle for message M, since, otherwise, B would have queried his signing oracle for this message, which would invalidate the forgery.

E2: This event comprises several subevents which we discuss below:

E2a: If A outputs a forgery of the form $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}_1, \mathsf{cert}, \sigma), \mathsf{pk}_i)$, where $i \neq 1$ so that $\mathsf{PV}(\mathsf{pk}_i, M, \sigma_p) = \mathtt{true}$, and no valid query (i, l, M) was made to oracle $O_\mathsf{PS}((skp_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, for $l \in \mathbb{N}$, it is immediate that A has not issued a signing query for $W_M\|r_M\|\mathsf{pk}_1$ to its signing oracle $\mathsf{O}_{\mathsf{S}_1}$, since, otherwise, B would have issued the same query to O_S.

E2b: Besides the case where A comes up with an forged DSS signature under sk_1, A may reuse an already issued proxy signature σ_p^* issued by user 1 for some message M^* for another message $M \neq M^*$ with either $M \in \omega^*$ or $M \notin \omega^*$ (with ω^* being the warrant corresponding to commitment $\mathcal{C}^*$ in certificate cert^*). Note that this is due to our construction, i.e., since we do not sign messages directly, but commitments to warrants incorporating hash values of messages and witnesses for committed messages, respectively. Further, note that in this case the witness W^* is fixed due to the reuse of a signature. If this subevent occurs, then, in case of $\mathsf{WHPSS} = \mathsf{WHPSS}_{\mathsf{PolyCommit}}$, A has either found

- second preimages in H, i.e., found M such that $H(M) = H(M^*)$ whereas $H(M^*)$ is the root of the warrant polynomial of ω^* corresponding to the witness W^* signed in σ_p^*, or
- another warrant polynomial $m'(X)$ with $m'(X) \neq m^*(X)$ so that $m'(\alpha) = m^*(\alpha)$. However, since α is unknown, finding such an $m'(X)$ requires breaking the polynomial binding property of $\mathsf{PolyCommit}_{\mathsf{Ped}}$, or

in case of $\mathsf{WHPSS} = \mathsf{WHPSS}_{\mathsf{VectorCommit}}$, A has either

- found second preimages for the hash values on the path from the leaf commitment to the root, where the path is fixed by the siblings contained in W^* in σ_p^*. W.l.o.g. it suffices to find a single second preimage for the parent node of the leaf commitment, or
- has broken the r-binding of the leaf commitment $\mathcal{C}^*$ contained in σ_p^*, i.e., has found O' such that $\mathsf{COpen}(\mathcal{C}^*, O') = M \neq M^*$.

E3: As in event **E2**, this event comprises several subevents which we discuss below:

E3a: A outputs a forgery of the form $(M, \sigma_p, \mathsf{pk}_1)$ with $\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathsf{pk}_j, \mathsf{cert}, \sigma)$ such that $\mathsf{ID}(\sigma_p) = j$ and $\mathsf{PV}(\mathsf{pk}_1, M, \sigma_p) = \mathtt{true}$. Since we require that for all ω with $(j, \omega, \mathcal{C}) \in DU$, $M \notin \omega$ it is immediate that A has not issued a signing query for $W_M \| r_M \| \mathsf{pk}_1$ to its signing oracle $\mathsf{O}_{\mathsf{S}_1}$ as, otherwise, B would have issued the same query to O_{S}.

E3b: Here, in analogy to the above event, it remains to argue that A may reuse an already issued proxy signature σ_p^* for proxy j with a new message M^* with $(j, \omega, \mathcal{C}) \in DU$, $M^* \notin \omega$. If this subevent occurs, then, in case of $\mathsf{WHPSS} = \mathsf{WHPSS}_{\mathsf{PolyCommit}}$ as well as $\mathsf{WHPSS} = \mathsf{WHPSS}_{\mathsf{VectorCommit}}$, the argumentation is exactly as in **E2b**.

Adversary C: The setup and environment for adversary C is identical to the one of adversary B, but C is an adversary against DSS only. After running the simulation, A eventually outputs an attempted forgery of the form $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}', \mathsf{cert}, \sigma), \mathsf{pk}_1)$. C will either abort or output a forgery of the form $(\mathsf{pk}_1, \mathsf{pk}', \mathcal{C}\|1\|\mathsf{pk}', \mathsf{pk}_1\|W_M\|r_M, M, \mathsf{cert}, \sigma)$.

Subsequently, we analyze adversary C. The view of A in this simulation is identical to that in the real game and if one of the given events occurs, then C has won the game.

E4: Note that if A outputs a valid forgery $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}', \mathsf{cert}, \sigma), \mathsf{pk}_1)$, then cert is a signature on $\mathcal{C}\|1\|\mathsf{pk}'$ under sk_1 and σ is a signature on $W_M\|r_M\|\mathsf{pk}_1$ under sk'. Clearly, if A outputs a valid forgery, then $\mathcal{C}\|1\|\mathsf{pk}'$ has not been issued to the signing oracle, since this event is defined such that A has not run the self delegation for pk'.

Adversary D: Adversary D is against DSS. It gets a public key pk and has access to a signing oracle under sk. Then, D simulates the environment for adversary A as follows: D selects a random index $t \in \{1, \ldots, q_s\}$ (one of the q_s self-delegation queries that will be run by A), initializes the counter and the arrays as above. However, in contrast to adversaries B, C and D generates the key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$ by himself and runs A on input pk_1. We highlight the differences between the simulated environments of B and C and the simulated environment of D below (essentially, the differences are in handling self-delegation queries, the delivery of proxy keys and proxy signing queries):

1. Identical to B.
2. Instead of using the signing oracle $\mathsf{O}_{\mathsf{S}}(\mathsf{sk}_1, \cdot)$, D can use sk_1 to produce the signatures cert.

3. Since D knows sk_1, it also stores sk_1 into skp_i.
4. If A requests user 1 to run the designation protocol with itself for message space ω, then: For every query except for the t'th self-delegation query made by A, B runs K to obtain $(\mathsf{pk}_1', \mathsf{sk}_1')$, and produces a signature cert on $\mathcal{C}\|1\|\mathsf{pk}_1'$. It then forwards $(\omega, \mathcal{C}, s, \mathsf{cert})$ to A and stores $(\mathsf{sk}_1', \mathsf{pk}_1, s, \mathcal{C}, 1, \mathsf{pk}_1', \omega,\ \mathsf{cert})$ in the last unoccupied position of skp_1. For the t'th self-delegation query, D produces a signature cert on the message $\mathcal{C}\|1\|\mathsf{pk}$ using sk_1 (note, pk is the key that was given as input to D). Then, D forwards $(\omega, \mathcal{C}, s, \mathsf{cert})$ to A.
5. If A requests to see $\mathsf{skp}_1[i]$ for some $i \neq t$, then let $(\mathsf{sk}_1', \mathsf{pk}_1, s, \mathcal{C}, 1, \mathsf{pk}_1', \omega, \mathsf{cert})$ be the i'th entry in skp_1. Adversary D forwards $(\mathsf{sk}_1', \mathsf{pk}_1, s, \mathcal{C}, 1, \mathsf{pk}_1', \omega, \mathsf{cert})$ to A and sets CS to $CS \cup \omega$. In case of $i = t$, D aborts its execution.
6. If A queries its oracle $\mathsf{O}_{\mathsf{S}_1}(\mathsf{sk}_1, \cdot)$ with a message M, B produces a signature σ on M using sk_1 and returns σ to A.
7. If A makes a query (i, l, M), where $i \in [n], l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its proxy signing oracle $\mathsf{O}_{\mathsf{PS}}((\mathsf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, D responds as follows. If $i = 1$ and $l \neq t$ or $i \neq 1$ and $\mathsf{skp}_1[l]$ is not defined, then return $\perp$ to A. If $i = 1$ and $l \neq t$ or $i \neq 1$ and $\mathsf{skp}_1[l]$ is defined, then return $\mathsf{PS}(\mathsf{skp}_1[l], M)$ to A. If $i = 1$ and $l = t$, then send a query for $(W_M\|r_M\|\mathsf{pk}_i)$ to the signing oracle to obtain a signature σ under sk. The proxy signature that D returns to A is $(1, \mathcal{C}, W_i, r_i, \mathsf{pk}, \mathsf{cert}, \sigma)$.

Eventually, A outputs an attempted forgery $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}, \mathsf{cert}, \sigma), \mathsf{pk}_1)$, C will then output a forgery of the form $(\mathsf{pk}_1, \mathsf{pk}, \mathcal{C}\|1\|\mathsf{pk}, W_M\|r_M\|\mathsf{pk}_1, M, \mathsf{cert}, \sigma)$ and aborts otherwise. Subsequently, we analyze adversary D. The view of A in this simulation is identical to that in the real game and if one of the given events occurs, then D has won the game.

E5: Note that if A outputs a valid forgery $(M, (1, \mathcal{C}, W_M, r_M, \mathsf{pk}, \mathsf{cert}, \sigma), \mathsf{pk}_1)$, then cert is a signature on $\mathcal{C}\|1\|\mathsf{pk}$ and σ is a signature on $W_M\|r_M\|\mathsf{pk}_1$ under sk_1 and sk, respectively. Clearly, if A outputs a valid forgery, then A did not make the t-th self-delegation query $(1, t, M)$ for message M and, consequently, D did not query the signing oracle under sk. The probability that this happens is $1/q_s$ of the probability that **E5** occurs.

Taking all this together gives us the desired result and Theorems 2 and 4 follow. □

A.2 Proof of Theorems 3 and 5

For our game we define the following event:

E6: A outputs a warrant ω' such that $\omega' = \omega^*$, whereas ω^* is of size c and A has issued at most $c - 1$ queries for proxy signatures of user 1 delegated by user i under skp^*.

In order to deal with this event **E6**, we introduce an adversary E. Our goal is to show that $\mathsf{Pr}[\mathbf{E6}]$ is less or equal to the advantage of adversary E. Consequently, since this advantage is negligible by our assumptions, it follows that the advantage of adversary A is negligible. Consequently, if adversary A produces valid forgeries with non-negligible probability, i.e., the event happens, we obtain our desired contradiction to the hiding of $\mathsf{WCommit}$, the security of the PRG f or the security of the hash function H.

Let A be an efficient adversary against the privacy of WHPSS. We construct an adversary E against $\mathsf{WCommit}$, the PRG f or the hash function H. Adversary E simulates the challenger in the game of Definition 7 interacting with adversary A, where the keys of user 1 are $(\mathsf{sk}_1, \mathsf{pk}_1)$. Adversary E works as follows: First E initializes a counter $n = 1$ that keeps track of the number of users, generates a key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$, creates an empty array skp_1. Note that in this context it is not necessary to keep track of the warrants by using the sets DU and CS. Also note that A and E have access to the common reference string P, if necessary. It then runs A on input pk_1, handling A's requests and answering A's queries as the simulation conducted by B in proof of Theorems 2 and 4 (the only difference is that E does not need to access a signing oracle for sk_1, but can compute signatures under key sk_1 on its own). Note that in contrast to the above setting, A, at some point, signals that it is ready to proceed to phase 2 by sending (i,c) to E. E then produces the proxy signing key skp^* by choosing ω^* of size c and running $(\mathsf{D}(\mathsf{pk}_1,\mathsf{sk}_1,i,\mathsf{pk}_i,\omega^*),\mathsf{P}(\mathsf{pk}_i,\mathsf{sk}_i,\mathsf{pk}_1))$ and storing it to array skp'_i, where in case of $\mathsf{WHPSS} = \mathsf{WHPSS}_{\mathsf{PolyCommit}}$ we require $c \leq t$. More precisely, E constructs the warrant by choosing c random messages from some efficiently samplable space that is bounded by some polynomial. [2]

Now, we describe how E answers A's queries for proxy signatures under skp^*:

– On receiving a query, E chooses at random an unused index l and corresponding message M_l from ω^* and runs $\sigma^*_{p_l} = \mathsf{PS}(\mathsf{skp}^*, M_l)$ and returns it to A (note that A has no access to skp^* and is consequently not aware of ω^*).

Additionally, A is allowed to issue queries as in the first phase. Eventually, A outputs a warrant ω'. The view of A in this simulation is identical to that in the real game and if the event occurs, then E has won the game. Let us now analyze adversary E:

E6: Let us first analyze the case of $\mathsf{WHPSS} = \mathsf{WHPSS}_{\mathsf{PolyCommit}}$ and describe what A knows:

– A knows the commitment $\mathcal{C}^*$ to the warrant polynomial $m^*(X)$ of degree c, $c-1$ roots $H(M_1),\ldots,H(M_{c-1})$, the corresponding witnesses $W^*_1,\ldots,W^*_{c-1}$ and $r^*_1,\ldots,r^*_{c-1}$. Note that from the $c-1$ evaluations r^*_i's of the polynomial $r^*(X)$ of degree c, A is not able to interpolate $r^*(X)$ and consequently the commitment $\mathcal{C}^*$ remains unconditionally hiding. This means that A can try to construct $m'(X)$ for all messages in the message space (which is efficiently samplable and polynomially bounded) and will still not be able to figure out which polynomial is hidden in $\mathcal{C}^*$.
Alternatively, A can try guessing the coefficients of the random polynomial $r^*(X)$ produced by the PRG f (thereby A could use the $c-1$ known evaluations of $r^*(X)$) to break the hiding property of the polynomial commitment scheme and, consequently, to determine the remaining message in ω^* by brute-forcing $\mathbb{M}$.
Clearly, if A outputs ω' such that $\omega' = \omega^*$ then it must have either broken the unconditionally hiding property of $\mathsf{PolyCommit}_{\mathsf{Ped}}$ or the security (output indistinguishability from random) of the PRG f.

and now we investigate the case of $\mathsf{WHPSS} = \mathsf{WHPSS}_{\mathsf{VectorCommit}}$ and describe what A knows:

– A knows the commitment $\mathcal{C}^*$ to the warrant $\omega^* = (M_1,\ldots,M_c)$, w.l.o.g. the messages $M_1,\ldots,M_{c-1}$ and $c-1$ witnesses (authentication paths) W_{M_l} and corresponding openings (M_l, r_l) to the leaf commitments. Firstly, let us assume that c is even and, thus, A knows all c leaf commitments $\mathcal{C}_l$, which are included in one of the witnesses W_{M_l}. Let $\mathcal{C}_c$ be the remaining commitment, which has not been opened yet. Consequently, A needs to find the unknown pair (M_c, r_c) corresponding to $\mathcal{C}_c$. Secondly, in case of odd c, A may not even know $\mathcal{C}_c$, but only the parent hash value thereof obtained from one of the witnesses. Clearly, this would require A to additionally reconstruct $\mathcal{C}_c$ from the parent hash value.
Now, if A outputs ω' such that $\omega' = \omega^*$ then it must have either broken the unconditionally hiding property of the leaf commitment scheme or the security (output indistinguishability from random) of the PRG f. In the second case, A must also have broken the preimage resistance of the hash function used in the Merkle tree.

This gives us the desired result and Theorems 3 and 5 follow. □

[2] Note that this restriction to an efficiently samplable message space makes our privacy guarantees even stronger, since even if the adversary is able to efficiently test any message from the space, it will not gain any advantage in winning the game.