Anonymous, Timed and Revocable Proxy Signatures

Ghada Almashaqbeh[1(✉)] and Anca Nitulescu[2]

[1] University of Connecticut, Storrs, CT, USA
ghada@uconn.edu
[2] IOG, Paris, France
anca.nitulescu@iohk.io

Abstract. A proxy signature enables a party to delegate her signing power to another. This is useful in practice to achieve goals related to robustness, crowd-sourcing, and workload sharing. Such applications, especially in the blockchain model, usually require delegation to satisfy several properties, including time bounds, anonymity, revocability, and policy enforcement. Despite the large amount of work on proxy signatures in the literature, none of the existing schemes satisfy all these properties; even there is no unified formal notion that captures them.

In this work, we close this gap and propose RelaySchnorr, an anonymous, timed, and revocable proxy signature scheme. We achieve this in two steps: First, we introduce a tokenizable digital signature based on Schnorr signature allowing for secure distribution of signing tokens. Second, we utilize a public bulletin board, instantiated as a blockchain, and timelock encryption to support: (1) one-time usage of the signing tokens by tracking tokens used so far based on unique values associated to them, (2) timed delegation so that a proxy signer cannot sign outside a given period, and (3) delegation revocation allowing the original signer to end a delegation earlier than provisioned. All of these are done in a decentralized and anonymous way so that no one can tell that someone else signed on behalf of the original signer or even that a delegation took place. We define a formal notion for proxy signatures capturing all these properties, and prove that our construction realizes this notion. We also discuss several design considerations addressing issues related to deployment in practice.

1 Introduction

Proxy signatures allow one user (the original signer) to delegate their signing right to another party (the proxy signer). The proxy signer can generate signatures that are verified using the original signer's certified public key.[1] Proxy signatures are useful in many applications related to distributed systems, e-cash using smart cards, grid computing, and workload sharing [16,31,33,38]. For example, Alice can let her assistant (Bob) reply to (and sign) emails on her

[1] This is not to be confused with proxy re-signatures [3,7], in which Alice gives a trusted third party a secret key $sk_{b\to a}$ that is used to transform Bob's signature into Alice's signature. Our focus in this work is on signature delegation.

N. Mouha and N. Nikiforakis (Eds.): ISC 2024, LNCS 15257, pp. 23–43, 2025.
https://doi.org/10.1007/978-3-031-75757-0_2

behalf while on a vacation, or simply she can share the workload of handling emails with Bob. Anonymous delegation guarantees that no one can tell that the task was delegated. Alice may further limit the delegation rights to a certain task or period of time, and may retain the ability to revoke the delegation at any moment of her choice.

The concept of proxy signatures was first introduced in [30], where several types of delegation were presented: full delegation—the original and proxy signers share the same secret key, partial delegation—the original signer generates a delegation key for the proxy signer to use, and delegation by warrant—the original signer specifies a policy that restricts which messages the proxy signer can sign. Since then, a large number of followup works emerged analyzing security and efficiency of older schemes, and devising new constructions, e.g., [4,14,23–25,31,39]. Other foundational works focused on formulating and unifying the security requirements of proxy signatures, and strengthening the adversary model [9,29,35]. Furthermore, several works focused on extending proxy signatures to handle different settings and support new features, such as threshold proxy signatures [22,37], blind proxy signatures [42], and anonymous proxy signatures [17].

Motivation. Our motivation stems from recent advances in distributed systems and Web 3.0 applications, and their need for delegation of signing rights. The continuous emergence of such applications raises the question of whether we can realize many of the robustness, user-control, and flexible-configuration features (facilitated by trusted banks/financial institutions) in the trustless blockchain model. This model also introduced new needs that did not exist before, such as managing hot and old wallets and targeted attacks against consensus. Having cryptographic schemes that cover various properties is valuable as these can always become off-the-shelf solutions for new needs arising in this active area. In this work, we identify the properties and capabilities that decentralized applications require from signature delegation, and how to realize them. Towards this, we discuss two motivating applications, and we explain how existing solutions are not suitable as there is no single scheme that supports *all* the required properties.

Application 1: Decentralized Finance (DeFi) and Wallet Management. In DeFi, blockchains are used to facilitate financial services. There is always the question of whether DeFi can replace traditional banking systems. For example, can Alice allow a family member to spend currency from her Ethereum account in a controlled way without giving away her secret key? (in a similar way to issuing a credit card to a family member while the original account owner controls the credit limit and activation period of the new card.) Delegated signatures can easily enable that: Alice delegates signing rights to her sister Eve, under a *policy* ensuring that, e.g., only transactions with capped values can be issued, and that delegation is valid only for a *given period*. Alice can also end, or *revoke*, the delegation earlier if desired. Given the decentralized nature of blockchains, Alice wants to do all of that in a *decentralized* way.

Another use case is related to cold and hot cryptocurrency wallets. Cold wallets are used to store most of the funds and are not connected to the Internet. It happens that the hot wallet (e.g., a mobile application) may need more funds

for a particular activity than what was originally anticipated. Transferring funds between the two wallets requires the cold wallet to be connected to the network, which is risky. By delegating the signing capability to the hot wallet, it can *non-interactively* transfer funds out of the cold wallet, reducing security risks.

Application 2: System Robustness. Another important application is related to system robustness and addressing targeted attacks. Take Byzantine agreement-based consensus as an example. For each round, a committee will agree on the next block. A party, say Alice, can designate a few other parties as backups to sign on her behalf. If Alice's machine is down, Bob can sign on Alice's behalf after a preset timeout. So, even if Alice is a victim of a targeted attack, Bob will do the work until she recovers. Here *anonymity* is a key; backups must be anonymous to avoid targeted attacks against them as part of attacking Alice. The same analogy can be applied to any system in which designated parties must be available to perform particular functionalities.

These applications outline several desired properties. *Delegation anonymity* hides that a delegation took place and the identities of the proxies, ensuring that to an outsider everything appears to be done by the original signer. Also, delegation should be of an *ephemeral nature* so it can be exercised during a preset time period. Another feature is *revocability*, which could be automatic when the delegation period is over, or on-demand due to unforeseen circumstances. Moreover, *policy enforcement* allows restricting the proxy signer's power, e.g., signing messages that belong to a certain class. Furthermore, *decentralization and non-interactivity* are important features especially for large-scale distributed systems. That is, all delegation information are generated by the original signer and can be sent to the proxy signer in one shot; no further interaction with the original signer and no trusted third party are needed when the proxy generates signatures. This promotes scalability and agrees with the spirit of delegation—the original signer can go offline once the delegation is created.

Despite the large amount of work around signature delegation, there is no single proxy signature scheme that achieves all the properties mentioned above, and there is no formal notion that covers all these properties. Full delegation, by giving away the signing key, offers anonymity but at the price of losing control over the delegation. Many schemes allow some fine-grained control [14,20,30], but violate anonymity. Others [4,17] support controlled and anonymous delegation, but without any revocation capability or timed notion. At the same time, schemes that support timed delegation and/or revocation [27,28,36,41] either do not offer anonymity, require interaction between the original and proxy signers, rely on trusted/semi-trusted third party, or do not have formal security analysis.

These observations raise the following question: *Can we construct a decentralized and non-interactive proxy signature scheme that is anonymous, timed and revocable? and how to formally define its security?*

1.1 Contributions

In this paper, we answer this question in the affirmative by defining a formal notion for anonymous, timed and revocable proxy signatures, and constructing

a proxy signature scheme, called RelaySchnorr, that satisfies all the properties discussed above.

Formal Modeling. Our notion builds on previous proxy signature definitions [9,17,29] and defines the additional properties that we require. We generalize the notion to produce generic delegation information instead of restricting these to be tokens or delegation keys. We also introduce a revocation algorithm covering automatic and on-demand revocability. For policy enforcement, we view it as two parts: a policy over time encapsulating the delegation period, and a policy over the message specifying which messages can be signed. To support delegation anonymity, verification of all signatures is done under the original signer's public key without involving the proxy signers' identities. We formally define correctness and security, where the latter covers unforgeability under chosen message attacks, anonymity, revocability, and policy enforcement. We note that none of the existing definitions covered all these properties at once, and none of them defined a time policy or revocation.

Construction. We introduce a new proxy signature scheme, RelaySchnorr, that realizes our notion. RelaySchnorr combines Schnorr signature, timelock encryption, and a public bulletin board to support token-based delegation that is anonymous and revocable, and enforces time and message policies.[2] We introduce a *one-time tokenizable digital signature scheme* based on Schnorr signatures. This is done via a *two-layered approach*: the first layer produces a token, while the second layer produces a signature over the intended message using this token. The original signer can produce tokens on her own and communicate them securely to the proxy signer. Signature verification is done against the public key of the original signer, and the signature structure is identical whether it is generated by the original or proxy signer. In terms of size, a signature consists of four field elements and one group element, so the cost of delegation consists of one group element and two field elements compared to original Schnorr signatures.

We enforce *one-time use* of a token by publishing its unique value (i.e., a unique random element k) on the bulletin board. Any signature with an already published k value will be rejected, and upon accepting a valid signature, the verifier publishes the corresponding k value on the board, preventing the proxy signer from reusing a token. This differs from conventional one-time signatures, where if a signer signs more than one message her signing key will be revealed. A proxy signer would attempt to reveal the original signer's signing key, and hence the conventional notion does not work in our setting. Our approach does not reveal the key even if a proxy signer (locally) uses a token to sign several messages, and still only one of these signatures will be accepted. As noted, verifiers are trusted to publish the k values of the signatures they accept. In blockchain applications (our main target) signed transactions are verified by the miners/validators, so consensus honest majority guarantees this behavior.

[2] We assume a secure bulletin board that is an append-only, publicly-accessible log instantiated in a decentralized way as a blockchain maintained by a set of miners (we refer to the miners as validators). Any secure bulletin board (that satisfies persistence and liveness) can be used whether it is based on proof-of-work, proof-of-stake, etc.

For the *timed delegation*, we achieve that in a decentralized way without involving a time server or public warrants that compromise anonymity. We utilize the bulletin board along with timelock encryption [19]. In timelock encryption, a ciphertext is locked to a time ρ so that when time ρ comes, some public information will become available allowing for decryption. To enforce a proxy signer to exercise the delegation within a time period $[\rho_a, \rho_b]$, where ρ is a round number from the board, the original signer encrypts the delegation tokens locked to time ρ_a and privately sends them to the proxy signer. Only at time ρ_a the proxy signer can access the tokens. To enforce the end of the period, or *automatic revocation of delegation*, the original signer encrypts all unique k values of these tokens in another ciphertext locked to time ρ_b and publishes that on the board. When time ρ_b comes, the board validators will decrypt and publish all unused k values preventing the proxy signer from using any unused tokens after ρ_b.

For *on-demand revocation*, we do that in a similar way to automatic revocation. The difference is that the original signer publishes the unused k values of the delegated tokens before time ρ_b. To the best of our knowledge, we are the first to support revocability and the timed notion in an anonymous and decentralized way. For *policy enforcement over messages*, we follow generic approaches [17,20] for two cases: public policy (using the warrant approach) and private one (using non-interactive zero knowledge proofs—NIZKs).

A few challenges arise when deploying our scheme in practice. Examples include synchronization issues of the board, denial of service (DoS) attacks, and anonymity concerns related to mass publication of k values during revocation. We discuss these and other challenges, along with solutions, in our construction.

Security. We formally prove security of RelaySchnorr based on our notion. Unforgeability relies on the unforgeability of Schnorr signatures in the random oracle model [34], and the Schnorr knowledge of exponent assumption [5,12]. Anonymity is achieved by having identical signature structure and behavior (i.e., with respect to any information published on the board) for the original and proxy signers. Revocability relies on the security of timelock encryption and the bulletin board. Policy enforcement relies on the security of digital signatures (for public warrants) or NIZKs (for private policies), as well as security of timelock encryption and the bulletin board.

Lastly, we note that the techniques we devise to support the timed/revocability notion could be of independent interest; they could be used to support these features for other cryptographic functionalities. Furthermore, the reliance on Schnorr signature, which is a widely studied and used cryptographic primitive, favors construction simplicity. This could also make it easier for our construction to be adopted in practice by systems that use Schnorr signatures (e.g., Bitcoin is awaiting the adoption of Schnorr signatures as proposed in BIP 340 [40]).

1.2 Related Work

We review existing proxy signature schemes showing that none of them support all the properties we aim to achieve. In the full version [2], we further review works on relevant notions and position our work with respect to these efforts.

Anonymity. Anonymity is usually not supported since the proxy signer's key is public and needed for verification, e.g., [30]. Fuchsbauer et al. [17,18] address this issue by unifying the notion of proxy and group signatures, and they consider traceability with a trusted authority holding a trapdoor that can compromise anonymity if needed. Beside introducing a centralized entity, this scheme does not support revocation or timed delegation. Functional signatures [11] allow deriving a secret key sk_f, from the original signer's key, so a proxy signer can sign a message m only if $f(m) = 1$ (where f represents the policy). Delegatable functional signatures [4] utilize signature malleability to allow delegation. Both notions support anonymity but not timed or revocable delegation.

Time-Bounded Delegation. The few works on timed notion for delegation are limited. Lu et al. [28] add the delegation period to the public warrant, and relies on a trusted server to issue a timestamp for each signature a proxy signer wants to generate. Sun et al. [36] adopt an interactive delegation process; a verifier asks the proxy signer to sign a message, and one of these parties generates a timestamp that the other verifies. Beside being interactive and involving a trusted server, these schemes violate anonymity and do not have formal security.

Revocation. Techniques for revoking delegation rights [30,41] are based on changing the original signer's key so all delegated signatures will be rejected, or on creating a public list of revoked proxy signers' keys. The scheme in [41] uses revocation epochs, where for each signature, the proxy signer generates a proof that her key is not on the epoch revocation list, so anonymity is not supported. Others [13,27,28] rely on a (semi-)trusted server, where the proxy signer must contact this server when generating a signature: the original signer updates the server with all revoked proxy signers to deny their requests. This approach introduces centralization and trust issues, which we avoid in our scheme.

Policy Enforcement. Warrants are used to enforce a policy over delegation, and are usually public—a verifier rejects any signature over a message that violates the warrant [9,29]. Private warrant approaches either rely on NIZKs to show that a signed message belongs to a hidden (committed) set of messages [21], on polynomial commitments to restrict the proxy signer to sign messages following a specific template [20], or on anonymous non-interactive credentials [14]. However, none of these schemes offer anonymity. Functional (delegatable) signatures [4,11] and traceable policy-based signatures [1] support private policy and proxy anonymity using NIZKs, but without any timed notion for the delegation (and the latter requires interaction between the proxy and the tracing authority). In terms of message policy enforcement, we use these generic approaches, while we leave traceability for future work as we do not to involve a trusted entity.

2 Preliminaries

We provide an overview of timelock encryption (TLE) that we utilize in our construction. More about the correctness, security, and candidate constructions of TLE, and an overview of Schnorr signatures (where we adopt the formulation from [32] that mitigates related key attacks), can be found in the full version [2].

Notation. We denote the natural numbers by $\mathbb{N}$, the integers by $\mathbb{Z}$, and the integers modulo some q by $\mathbb{Z}_q$. Elements of $\mathbb{Z}_q$ are lowercase, and elements of a multiplicative group $\mathbb{G}$ of order q generated by generator $G \in \mathbb{G}$ are uppercase. λ denotes the security parameter, pp denotes the public parameters, $\mathsf{negl}(\lambda)$ denotes a function negligible in λ, and PPT stands for probabilistic polynomial time.

Timelock Encryption (TLE). TLE enables encrypting messages towards a time ρ (which is a round number from the bulletin board) such that they can be decrypted only after that time. Thus, for each ρ, a round-related information π_ρ is published on the board to enable decryption. We adopt the definition in [19] with a more generalized time information production algorithm as shown below.

Definition 1 (Timelock Encryption (TLE)). *A Timelock encryption scheme $\mathcal{E}$ is a tuple of five* PPT *algorithms defined as follows:*

$\mathsf{TLE.Setup}(1^\lambda) \to (\mathsf{pp}, s)$**:** *Takes as input the security parameter λ, and outputs public parameters* pp *and a private key s.*[3]

$\mathsf{TLE.RoundBroadcast}(s, \rho) \to \pi_\rho$**:** *Takes as input the round number ρ and a private key s, and outputs the round-related decryption information π_ρ.*

$\mathsf{TLE.Enc}(\rho, m) \to (\mathsf{ct}_\rho, \tau)$**:** *Takes as input the round number ρ and a message m, and outputs a round-encrypted ciphertext ct_ρ, and trapdoor τ for pre-opening.*

$\mathsf{TLE.Dec}(\pi_\rho, \mathsf{ct}_\rho) \to m'$**:** *Takes as input the round-related decryption information π_ρ and a ciphertext ct_ρ, and outputs a message m'.*

$\mathsf{TLE.PreOpen}(\mathsf{ct}_\rho, \tau) \to m'$**:** *Takes as input a ciphertext ct_ρ and a trapdoor τ, and outputs a message m'.*

We use TLE in a blackbox way by invoking the algorithms defined above. We leave any details of, e.g., model and security of the blockchain and time information, to the concrete instantiation (which we require to be fully decentralized and publicly verifiable). We briefly discuss TLE correctness and security, and TLE instantiations in the full version [2].

3 Definitions

In this section, we formulate a notion for anonymous, timed and revocable proxy signature scheme. We build on previous definitions for proxy signatures [9,17,29] and extend them to cover the additional properties we require.

Definition 2. *An anonymous, timed and revocable proxy signature scheme Σ is a tuple of seven* PPT *algorithms defined as follows:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$**:** *Takes the security parameter λ as input, and outputs a set of public parameters* pp*.*[3]

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$**:** *Takes the security parameter λ as input, and outputs a signing key* sk *and a verification key* vk*.*

$\mathsf{Sign}(\mathsf{sk}, m, \mathsf{policy}) \to \sigma$**:** *Takes the signing key* sk*, a message m, and a policy* policy *as inputs, and outputs a signature σ over m.*

[3] The public parameters pp are implicitly input to all subsequent algorithms.

$\mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec}) \rightarrow (\mathsf{degInfo}, \mathsf{rk})$**:** *Takes as inputs the signing and verification keys* $(\mathsf{sk}, \mathsf{vk})$*, and delegation specifications* degspec *(i.e., any auxiliary information and the policies over the time period and the messages that can be signed). It outputs delegation information* degInfo *and a revocation key* rk*.*

$\mathsf{DegSign}(m, \mathsf{degInfo}) \rightarrow \sigma$**:** *Takes a message* m *and the delegation information* degInfo *as inputs. It outputs a signature* σ *over* m*.*

$\mathsf{Revoke}(\mathsf{degInfo}, \mathsf{rk}, \mathsf{revState}[\mathsf{vk}]) \rightarrow \mathsf{revState}[\mathsf{vk}]'$**:** *Takes* degInfo*, the revocation key* rk*, and the revocation state associated to* vk *as inputs, and outputs an updated revocation state* $\mathsf{revState}[\mathsf{vk}]'$*.*

$\mathsf{Verify}(\mathsf{vk}, m, \sigma, \mathsf{revState}[\mathsf{vk}]) \rightarrow 1/0$**:** *Takes as inputs the verification key* vk*, a message* m*, a signature* σ *over* m*, and the revocation state* revState *of* vk*. It outputs 1 if the signature is accepted, and 0 otherwise.*

We require the scheme Σ *to satisfy the following properties: correctness, existential unforgeability under chosen message attacks, anonymity, revocability, and policy enforcement, which we define below.*

As shown above, the inputs for Sign include the policy, which we view to be composed of two sub-policies: policy_m determines which messages can be signed, and policy_t determines the time period so that a signature produced and verified outside this period will be rejected. Although the latter is needed for the timed delegation but we include it here to satisfy anonymity. That is, all signatures whether produced by the original or proxy signer will have the same structure. The term policy is general and could be extended to include additional polices if desired. Also, policy is configured by the original signer; the proxy signer cannot change it. For delegation, Delegate takes as input the delegation specifications degspec, which includes all auxiliary information needed for delegation configuration, such as number of signing tokens in token-based schemes, and the policy policy to be enforced. The produced degInfo that will be sent to the proxy signer will include the same policy found in degspec and all information a proxy signer needs to sign.

Our definition contains one verify algorithm used for all signatures whether produced by the original or proxy signer. Verify will reject any signature that does not satisfy the policy, and checks that the signature is not revoked (based on the revocation state revState associated with vk that is publicly available).[4]

Lastly, note that when Revoke is invoked, it updates the revocation state of vk to which the delegation is associated. This update is simply a concatenation operation that appends the new revocation information. Furthermore, this is a revocation of a delegation so the proxy signature cannot exercise the delegation anymore. It is not about revoking signatures that were already accepted.

[4] In our construction, Verify takes the current revocation state of vk (retrieved from the bulletin board) as input; it uses this state to check the freshness of the signature and then updates this state to include the unique value of the accepted signature to enforce one-time usage. One may argue that the definition of Verify should output an updated state revState to reflect that. However, we did not incorporate that to keep the definition general so it can be used by future constructions that may not rely on a bulletin board to enforce the one-time property.

Now we define the properties listed in Definition 2. We use code-based games [6] to formulate our security notions; an experiment $\mathsf{Exp}^{\mathsf{sec}}_{\Sigma,\mathcal{A}}$ is played with respect to a security notion sec and an adversary $\mathcal{A}$ against a scheme Σ. Furthermore, and similar to [8], we define the following helper functions that are not available in the actual scheme, but only for the challenger in the experiments.

$\mathsf{SimSetup}(1^\lambda) \rightarrow (\mathsf{pp}, \mathsf{td})$: Receives the security parameter λ, and outputs the public parameters pp which distribution is computationally indistinguishable from what is produced by Setup, and a trapdoor td.

$\mathsf{ExtDeg}(\mathsf{td}, \sigma, \{\mathsf{degInfo}\}) \rightarrow 1/0$: Receives a trapdoor td, a valid signature σ, and a set of delegation information $\{\mathsf{degInfo}\}$. It returns 1 if any of the $\mathsf{degInfo}$ in this set was used to generate σ, and 0 otherwise.

As shown, ExtDeg determines if a forgery signature was generated via a delegation based on hidden values in the signature. We resort to the use of such help function, as in [8], since this check cannot be performed on the available public signature values without a special extractor, since that would break the anonymity property of the scheme.

Correctness. Informally, correctness implies that a signer holding a valid secret key or delegation information can always produce a valid signature σ over a message m such that Verify will accept that signature if: the signature verifies correctly against vk; it is not revoked based on the latest version of $\mathsf{revState}[\mathsf{vk}]$; and that it does not violate the specified policy.

For all λ, all $m \in \{0,1\}^*$, any policy $\mathsf{policy} = (\mathsf{policy}_m, \mathsf{policy}_t)$ such that m satisfies policy_m and the time of signing/verification is within policy_t, any delegation specifications $\mathsf{degspec}$ and delegation information for this specifications $\mathsf{degInfo}$ such that $\mathsf{policy} = \mathsf{degspec.policy} = \mathsf{degInfo.policy}$, and the latest public revocation state $\mathsf{revState}$ based on which the signature σ is not revoked, the following probability is 1:

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \sigma, \mathsf{revState}[\mathsf{vk}]) = 1 \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec}) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m, \mathsf{policy}) \vee \\ \sigma \leftarrow \mathsf{DegSign}(m, \mathsf{degInfo}) \end{array}\right]$$

Existential Unforgeability. This property states that no adversary can produce a valid signature without the knowledge of the signing key sk or a delegation information $\mathsf{degInfo}$ created with respect to $(\mathsf{sk}, \mathsf{vk})$. Formally, for all λ, all $m \in \{0,1\}^*$, and any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that $\Pr[\mathsf{Exp}^{\mathsf{ProxyEUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$, where $\mathsf{Exp}^{\mathsf{ProxyEUF\text{-}CMA}}_{\Sigma,\mathcal{A}}$ is the experiment of existential unforgeability under chosen message attacks defined in Fig. 1, and the probability is taken over all randomness used in the experiment.

We note the following in the description of $\mathsf{Exp}^{\mathsf{ProxyEUF\text{-}CMA}}_{\Sigma,\mathcal{A}}$. Checking if the forged signature has been produced using a delegation obtained through $O\mathsf{Delegate}$ is done by invoking ExtDeg over the signature and $\mathsf{L}_{\mathsf{deleg}}$ (with the

$\mathsf{Exp}^{\mathsf{ProxyEUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda)$

1 : $\mathsf{L}_{\mathsf{sign}} \leftarrow \varnothing, \mathsf{L}_{\mathsf{deleg}} \leftarrow \varnothing$
2 : $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SimSetup}(1^\lambda)$
3 : $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
4 : $O \leftarrow \{O\mathsf{Sign}, O\mathsf{Delegate}\}$
5 : $(m^*, \sigma^*) \leftarrow \mathcal{A}^O(\mathsf{vk})$
6 : **if** $m^* \in \mathsf{L}_{\mathsf{sign}} \ \vee \ \mathsf{ExtDeg}(\mathsf{td}, \sigma^*, \mathsf{L}_{\mathsf{deleg}}) = 1$
7 : **return** 0
8 : **if** $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*, \mathsf{revState}[\mathsf{vk}]) = 0$
9 : **return** 0
10 : **return** 1

$O\mathsf{Sign}(m, \mathsf{policy})$

1 : $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m, \mathsf{policy})$
2 : $\mathsf{L}_{\mathsf{sign}} \leftarrow \mathsf{L}_{\mathsf{sign}} \cup \{m\}$
3 : **return** σ

$O\mathsf{Delegate}(\mathsf{vk}, \mathsf{degspec})$

1 : $(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
2 : $\mathsf{L}_{\mathsf{deleg}} \leftarrow \mathsf{L}_{\mathsf{deleg}} \cup \{\mathsf{degInfo}\}$
3 : **return** $(\mathsf{degInfo}, \mathsf{rk})$

Fig. 1. Existential unforgeability under chosen message attacks.

$\mathsf{Exp}^{\mathsf{DegAnon}}_{\Sigma,\mathcal{A}}(\lambda)$

1 : $b \xleftarrow{\$} \{0,1\}$
2 : $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
4 : $O \leftarrow \{O\mathsf{Delegate}, O\mathsf{Sign}\}$
5 : $(m^*, \mathsf{degspec}) \leftarrow \mathcal{A}^O(\mathsf{vk})$
6 : $\bar{\sigma} \leftarrow \mathsf{Chal}_b(m^*, \mathsf{degspec})$
7 : $b^* \leftarrow \mathcal{A}^O(\bar{\sigma})$
8 : **if** $b^* = b$
9 : **return** 1
10 : **return** 0

$\mathsf{Chal}_b(m^*, \mathsf{degspec})$

1 : **if** b $= 0$
2 : $\sigma_0 \leftarrow \mathsf{Sign}(\mathsf{sk}, m^*, \mathsf{degspec.policy})$
3 : **if** b $= 1$
4 : $(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
5 : $\sigma_1 \leftarrow \mathsf{DegSign}(m^*, \mathsf{degInfo})$
6 : **return** σ_b

Fig. 2. Anonymity for delegation ($O\mathsf{Sign}$ and $O\mathsf{Delegate}$ are as defined in Fig. 1).

trapdoor τ) which will decide if this signature was produced using any of the degInfo in $\mathsf{L}_{\mathsf{deleg}}$. Note that this helper function is not part of the actual scheme, and that τ is secret. Thus, seeing a signature does not reveal that a signature was produced by the original or proxy signer, thus compromising anonymity.

Delegation Anonymity. This implies that the verifier, or any adversary, will not be able to infer any information about a delegation (one that he does not know its degInfo). That is, all signatures will appear as if they were produced by the original signer—they do not reveal anything about the identity of the proxy signers or even that there are delegations in the first place. Thus, all signatures are indistinguishable (in terms of structure and behavior) and all are verified against the original signer's verification key vk. Also, the produced degInfo is

$\mathsf{Exp}^{\mathsf{DegRev}}_{\Sigma,\mathcal{A}}(\lambda)$

$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SimSetup}(1^\lambda), (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$O \leftarrow \{O\mathsf{Sign}, O\mathsf{Delegate}\}$
$\mathsf{degspec} \leftarrow \mathcal{A}^O(\mathsf{vk})$
$(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
$\mathsf{revState}[\mathsf{vk}]' \leftarrow \mathsf{Revoke}(\mathsf{degInfo}, \mathsf{rk}, \mathsf{revState}[\mathsf{vk}])$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^O(\mathsf{vk}, \mathsf{degInfo})$
if $\mathsf{ExtDeg}(\mathsf{td}, \sigma^*, \mathsf{degInfo}) = 1 \ \wedge \ \mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*, \mathsf{revState}[\mathsf{vk}]') = 1$
  return 1
return 0

Fig. 3. Delegation revocation ($O\mathsf{Sign}$ and $O\mathsf{Delegate}$ are as defined in Fig. 1).

transmitted to the proxy signer over a secure channel, thus outside these two parties, no one will be able to tell that such information was produced.

Formally, for all λ, all $m \in \{0,1\}^*$, and any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that $\Pr[\mathsf{Exp}^{\mathsf{DegAnon}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where $\mathsf{Exp}^{\mathsf{DegAnon}}_{\Sigma,\mathcal{A}}$ is the experiment of delegation anonymity defined in Fig. 2, and the probability is taken over all randomness used in the experiment.

As shown in the figure, the adversary $\mathcal{A}$ will choose a message m^* and delegation specifications $\mathsf{degspec}$ (where the latter includes a policy policy). The challenger, based on the value of b, signs m^* using either delegation information $\mathsf{degInfo}$ generated based on $\mathsf{degspec}$, or the signing key sk (hence, no delegation) and returns the signature to $\mathcal{A}$. The adversary $\mathcal{A}$ is challenged to tell which method was used for signing.

Revocability. This implies that an adversary $\mathcal{A}$ cannot produce a valid signature that will convince the verifier using a revoked delegation. Formally, for all λ, all $m \in \{0,1\}^*$, and any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that $\Pr[\mathsf{Exp}^{\mathsf{DegRev}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$, where $\mathsf{Exp}^{\mathsf{DegRev}}_{\Sigma,\mathcal{A}}$ is the experiment of delegation revocation defined in Fig. 3, and the probability is taken over all randomness used in the experiment.

As shown in the figure, $\mathcal{A}$ is challenged to produce a valid signature (that will be accepted) using the revoked delegation. Thus, the game checks that indeed the signature σ^* is produced using the revoked $\mathsf{degInfo}$ as done before. This check is needed to rule out the following trivial attack: $\mathcal{A}$, who has access to $O\mathsf{Delegate}$, can produce a valid signature that verifies under vk using a different delegation from the one in the challenge that is not revoked, thus always winning the game.

Policy Enforcement. Informally, this implies that an adversary holding a valid delegation cannot produce a signature, that will be accepted, such that policy is not satisfied. This covers violating the policy over the message or the time.

$\mathsf{Exp}^{\mathsf{DegPolicy}}_{\Sigma,\mathcal{A}}(\lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$O \leftarrow \{O\mathsf{Sign}, O\mathsf{Delegate}\}$

$\mathsf{degspec} \leftarrow \mathcal{A}^O(\mathsf{vk})$

$(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$

$(m^*, \sigma^*) \leftarrow \mathcal{A}^O(\mathsf{vk}, \mathsf{degInfo})$

Parse $\mathsf{degspec.policy}$ as $(\mathsf{policy}_m, \mathsf{policy}_t)$

if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*, \mathsf{revState}[\mathsf{vk}]) = 1 \ \wedge \ (m^* \notin \mathsf{policy}_m \vee \mathsf{now} \notin \mathsf{policy}_t)$

  return 1

return 0

Fig. 4. Policy enforcement ($O\mathsf{Sign}$ and $O\mathsf{Delegate}$ are as defined in Fig. 1).

Formally, for all λ, all $m \in \{0,1\}^*$, and any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that $\Pr[\mathsf{Exp}^{\mathsf{DegPolicy}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$, where $\mathsf{Exp}^{\mathsf{DegPolicy}}_{\Sigma,\mathcal{A}}$ is the experiment of delegation policy enforcement defined in Fig. 4, and the probability is taken over all randomness used in the experiment. As shown in the figure, we use the variable now to refer to the current time, which is publicly accessible in the system. Thus, to check that the time policy is violated we check that now is outside the time period specified in policy_t. Also, the notion $m^* \notin \mathsf{policy}_m$ represents checking that m^* does not satisfy policy_m.

4 Construction

We present RelaySchnorr, an anonymous, timed and revocable proxy signature scheme that realizes the notion defined in the previous section. It relies on distributing one-time signing tokens to the proxy signers. Towards that, we introduce a tokenizable version of Schnorr signature, and employ a public bulletin board (an immutable, append-only log maintained by a set of validators with an honest majority, which can be instantiated as a blockchain), and timelock encryption to enforce the one-time use of signing tokens as well as the timed and revocable properties. The full construction is shown in Figs. 5 and 6. To simplify the discussion, we present our scheme with only the time policy, and we assume full synchrony with the bulletin board (i.e., any message that is sent appears immediately there). Toward the end of the section, we discuss enforcing a policy over the messages and handling synchrony of the board, as well as issues related to reducing the storage and information lookup on this board.

As mentioned before, the board state state is public and accessible by all parties. The time notion we use is in term of rounds derived from this board (in a similar way as done in blockchains). That is, a round is a block index from the board, which we denote as $\mathsf{state.round}$. We also use the notation $\mathsf{state}[\mathsf{vk}]$ to

access the revocation state associated with vk, and $\mathsf{state.roundInfo}(\rho)$ to access the TLE decryption information published on the board for round ρ.

One-Time Tokenizable Signature Scheme. This is based on Schnorr signatures, and done via a two-layered approach; layer 1 produces a token, while layer 2 produces a signature over m using the token.

For the signing algorithm, as shown in Fig. 5, we first generate a token using the signing key $\mathsf{sk} = x$, that is actually a Schnorr signature over a random element k with a secret randomness r. In particular, the signature requires computing $w = \mathsf{H}(k, X, R)$, where $R = G^r$. Looking ahead, the tuple (z, w, k) will be the token given to the proxy signer. To sign a message m, the original signer uses z as a secret key and produces another Schnorr signature over m with randomness e (note that the value z is also a random element in $\mathbb{Z}_p$). So this signature will be over the value $c = \mathsf{H}(m, Z, E)$, where $Z = G^z$. The output signature is $\sigma = (w, c, s, k, Z)$. The verification algorithm uses the public key $\mathsf{vk} = X$ to verify a signature by verifying the two layers of the Schnorr signature. As shown in Fig. 6, this is done by computing the randomness R and E and then verifying that the signed hashes, w and c, are indeed correct hashes based on the computed randomness.

The one-time property is enforced as follows. Each new signature contains a unique, freshly generated value k. When receiving a valid signature, the verifier checks that k is not on the board state associated to vk, namely $\mathsf{state}[\mathsf{vk}]$, and if so, the signature will be accepted and the verifier posts the k value on the board to be appended to $\mathsf{state}[\mathsf{vk}]$.[5] Any signature with an already-published k will be rejected. Looking ahead, this will enable one-time use of delegated tokens. As noted, verifiers are trusted to behave as prescribed by publishing the k values of the signatures they accept. In blockchain applications (our main target) signed transactions are verified by the miners who publish valid signed transactions (that include the k values) on the blockchain, so consensus honest majority guarantees this behavior (hence, it is not a trust in a single entity). Also, as shown, due to the reliance on the updated bulletin board state to enforce the one-time property, our construction is stateful (in practice, updating the state is done automatically in blockchain applications since accepting a valid transaction means that this transaction and its signature—including k—will be published on the board).[6]

To preserve delegation anonymity, the original signer must mimic the behavior of signatures produced by delegation (which we will see shortly). As shown in Fig. 5, every now and then the original signer will publish on the board a TLE ciphertext of a list of fresh and previously-used k values (locked to a future time). This is done to mimic having ct_b associated with a delegation (and so the automatic revocation). Also, every now and then, the original signer will gener-

[5] The verifier sends k to the board validators who will update the state $\mathsf{state}[\mathsf{vk}]$ when publishing a new block.

[6] Having a bulletin board does not introduce direct interaction between the original and proxy signers; an original signer produces $\mathsf{degInfo}$ and sends it in one shot to the proxy signer who can exercise delegation without any help from the original signer.

Let λ be a security parameter, S be the original signer, P be the proxy signer, and TLE be a timelock encryption scheme as defined in Definition 1. Construct an anonymous, timed and revocable proxy signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Delegate}, \mathsf{DegSign}, \mathsf{Revoke}, \mathsf{Verify})$ as follows:

$\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, set $\mathbb{G}$ to be a cyclic group of a prime order q with a generator $G \in \mathbb{G}$ and hash function $\mathsf{H} : \{0,1\}^* \times \mathbb{G}^2 \to \mathbb{Z}_q$, initialize $\mathsf{state} = \{\}$, and invoke $\mathsf{TLE.Setup}(1^\lambda)$. Output $\mathsf{pp} = (\mathsf{TLE.pp}, \mathsf{H}, \mathbb{G}, G, q, \mathsf{state})$.

$\mathsf{KeyGen}(1^\lambda)$: On input the security parameter λ, choose uniform $x \in \mathbb{Z}_q$, then compute $X = G^x$. Output the signing key $\mathsf{sk} = x$ and the verification key $\mathsf{vk} = X$.

$\mathsf{Sign}(\mathsf{sk}, m)$: On input the signing key $\mathsf{sk} = x$ and some message m, do:

1. Choose uniform $k, r, e \in \mathbb{Z}_q$, compute $R = G^r$, $E = G^e$
2. Compute $w = \mathsf{H}(k, X, R)$, $z = (r + wx) \mod q$, and $Z = G^z$
3. Compute $c = \mathsf{H}(m, Z, E)$ and $s = (e + cz) \mod q$ (if $z = 0$ or $s = 0$ start again with fresh r and e)
4. Output the signature $\sigma = (w, c, s, k, Z)$

Every now and then, S either (1) populates a set klist from the stored k values and fresh ones, encrypts it as $(\mathsf{ct}_b, \tau_b) = \mathsf{TLE.Enc}(\mathsf{klist}, \rho_b)$, where ρ_b is some future round number, and posts (ρ_b, ct_b) on the board (resulting in $\mathsf{state}'[\mathsf{vk}] = \mathsf{state}[\mathsf{vk}] \| (\rho_b, \mathsf{ct}_b)$), or (2) posts a fresh klist on the board (resulting in $\mathsf{state}'[\mathsf{vk}] = \mathsf{state}[\mathsf{vk}] \parallel \mathsf{klist}$).

$\mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$: On input the keypair $(\mathsf{sk} = x, \mathsf{vk} = X)$ and delegation specifications $\mathsf{degspec} = (u, [\rho_a, \rho_b])$, where $u \in \mathbb{N}$ and $[\rho_a, \rho_b]$ is the delegation period, do the following:

1. Set $\mathsf{klist} = \{\}$
2. Do the following for $i \in \{1, \ldots, u\}$:
   (a) Choose uniform $k_i, r_i \in \mathbb{Z}_q$
   (b) Compute $R_i = G^{r_i}$ and $w_i = \mathsf{H}(k_i, X, R_i)$
   (c) Compute $z_i = (r_i + w_i x) \mod q$ (if $z_i = 0$ start again with fresh r_i)
   (d) Set $t_i = (z_i, w_i, k_i)$ and $\mathsf{klist} = \mathsf{klist} \cup \{k_i\}$
3. Compute two ciphertexts: $(\mathsf{ct}_a, \tau_a) = \mathsf{TLE.Enc}(t_1 \| \cdots \| t_u, \rho_a)$ and $(\mathsf{ct}_b, \tau_b) = \mathsf{TLE.Enc}(\mathsf{klist}, \rho_b)$ (where τ_b is the revocation key rk).
4. Set $\mathsf{degInfo} = (\rho_a, \rho_b, \mathsf{ct}_a)$
5. Output $(\mathsf{degInfo}, \mathsf{ct}_b \| \tau_b)$

S stores ciphertext ct_b and trapdoor τ_b to be used for revocation if needed (τ_a is dropped as it is not needed), posts (ρ_b, ct_b) on the board (resulting in $\mathsf{state}'[\mathsf{vk}] = \mathsf{state}[\mathsf{vk}] \| (\rho_b, \mathsf{ct}_b)$), and sends $\mathsf{degInfo}$ to P.

Fig. 5. RelaySchnorr—continued in Fig. 6.

ate a fresh list of k values and publish the list itself on the board. This is done to mimic the mass publishing of k values associated with on-demand revocation.

Timed Delegation. To delegate signing, i.e., the Delegate algorithm in Fig. 5, the original signer chooses the delegation specifications including the number of signing tokens and the delegation period. The original signer uses her signing key $\mathsf{sk} = x$ to generate u fresh signing tokens, denoted as $t_1, \ldots, t_u$. Each of

$\mathsf{DegSign}(m, \mathsf{degInfo})$: On input a message m and delegation information $\mathsf{degInfo}$, P does the following (let $\rho_{now} = \mathsf{state.round}$ be the current round number):

1. If $\rho_{now} < \rho_a$ or $\rho_{now} > \rho_b$, then do nothing
2. If $\rho_a \leq \rho_{now} \leq \rho_b$, then:
   (a) If $\mathsf{degInfo} = (\rho_a, \rho_b, \mathsf{ct}_a)$, then retrieve π_{ρ_a} from the board ($\pi_{\rho_a} = \mathsf{state.roundInfo}(\rho_a)$) and set $\mathsf{degInfo} = (\rho_a, \rho_b, \mathsf{TLE.Dec}(\pi_{\rho_a}, \mathsf{ct}_a))$
   (b) Pick an unused signing token $t = (z, w, k)$ from $\mathsf{degInfo}$
   (c) Compute $Z = G^z$
   (d) Choose uniform $e \in \mathbb{Z}_q$ and compute $E = G^e$
   (e) Compute $c = \mathsf{H}(m, Z, E)$, and $s = e + cz \mod q$ (if $s = 0$ start again with a fresh e)
   (f) Output the signature $\sigma = (w, c, s, k, Z)$

$\mathsf{Revoke}(\mathsf{degInfo}, \mathsf{rk}, \mathsf{state}[\mathsf{vk}])$: On input $\mathsf{degInfo} = (\rho_b, \mathsf{ct}_b)$, revocation key rk, and revocation state $\mathsf{state}[\mathsf{vk}]$, do (let $\rho_{now} = \mathsf{state.round}$ be the current round number):

1. If $\rho_{now} \geq \rho_b$, then retrieve π_{ρ_b} from the board ($\pi_{\rho_b} = \mathsf{state.roundInfo}(\rho_b)$) and compute $\mathsf{klist} = \mathsf{TLE.Dec}(\pi_{\rho_b}, \mathsf{ct}_b)$
2. If $\rho_{now} < \rho_b$, then use $\mathsf{rk} = \tau_b$ to compute $\mathsf{klist} = \mathsf{TLE.PreOpen}(\mathsf{ct}_b, \tau_b)$
3. Add all k values such that $k \in \mathsf{klist} \ \wedge k \notin \mathsf{state}[\mathsf{vk}]$ to the board state $\mathsf{state}[\mathsf{vk}]$ associated with vk resulting in an updated state $\mathsf{state}[\mathsf{vk}]'$.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma = (w, c, s, k, Z), \mathsf{revState} = \mathsf{state}[\mathsf{vk}])$: On input the verification key $\mathsf{vk} = X$, the message m, signature $\sigma = (w, c, s, k, Z)$ over m, and the revocation state $\mathsf{state}[\mathsf{vk}]$, if $k \in \mathsf{state}[\mathsf{vk}]$, then output 0. Else, add k to $\mathsf{state}[\mathsf{vk}]$ (resulting in $\mathsf{state}'[\mathsf{vk}] = \mathsf{state}[\mathsf{vk}] \parallel k$) and do the following:

1. Compute $R = Z \cdot X^{-w}$ and $E = G^s \cdot Z^{-c}$
2. Output 1 if and only if $w = \mathsf{H}(k, X, R) \ \wedge \ c = \mathsf{H}(m, Z, E)$.

Fig. 6. RelaySchnorr (cont.).

these tokens contains z (first layer Schnorr signature over a fresh k), and the corresponding w and k values.

Our goal is to enforce a time period over the delegation in a decentralized way and without violating anonymity. To do that, we utilize a recent notion of timelock encryption TLE (defined in Sect. 2) in the blockchain model (again, we view the bulletin board as a blockchain). We represent the time period $[a, b]$ in terms of block indices, or rounds, covering the intended period. That is, this period will be $[\rho_a, \rho_b]$, where ρ_a (respectively ρ_b) is the round number during which the block with index a (respectively index b) is mined. To force a proxy signer to use the signing tokens only during $[\rho_a, \rho_b]$, we propose the following. The original signer uses TLE to encrypt the tokens in a ciphertext ct_a such that when ρ_a comes, and so the decryption information π_{ρ_a} becomes publicly available, the proxy signer can decrypt ct_a to retrieve the tokens. To enforce the end of the period, recall that any signature with a k value that appears on the board state state will be rejected. Thus, the original signer uses TLE to produce another ciphertext ct_b for time ρ_b encrypting klist (the list of k values of the delegated tokens) and posts ct_b on the board. When time ρ_b comes, and so π_{ρ_b}

becomes publicly available, the board validators will decrypt ct_b and publish all unused k values in klist on the board (this is included under Revoke in Fig. 6). This will prevent the proxy signer from using the unused tokens after time ρ_b.

The original signer stores ct_b and its secret trapdoor τ_b that can be used for early revocation (if needed) as we explain shortly. He then sends the delegation information $\mathsf{degInfo} = (\rho_a, \rho_b, \mathsf{ct}_a)$ to the proxy signer over a secure channel since this is secret information, and posts (ρ_b, ct_b) on the board.

Delegated Signing. As shown in Fig. 6, at time ρ_a, the proxy signer decrypts ct_a using the decryption information π_{ρ_a} that will become available on the board at that time. This will reveal $\mathsf{degInfo}$ containing the signing tokens. The proxy signer can use any of these tokens to sign a message m as follows: it chooses a random e and computes a signature using any of the unused (k, w, z) in $\mathsf{degInfo}$, which produces the second layer Schnorr signature. This signature has the same structure as the signatures that the original signer would produce, and will be verified using the same Verify algorithm against the original signer's verification key $\mathsf{vk} = X$, thus preserving anonymity of the proxy signer.

Revocation. We support decentralized and anonymous two types of revocation: automatic, when the delegation period is over enforced by the timed property discussed above, and on-demand allowing the original signer to end the delegation before time ρ_b. Both are done by decrypting ct_b and publishing all unused k values on the board, preventing the proxy signer from using the tokens tied to them. The difference is that for automatic revocation, decryption is done using π_{ρ_b} that will become publicly available at time ρ_b. While for on-demand revocation, that only the original signer can execute, the trapdoor τ_b of $\mathsf{degInfo}$ (in particular ct_b) is used to $\mathsf{PreOpen}$ ct_b.

As mentioned in Sect. 2, we employ a decentralized TLE scheme in our construction. The decryption information is produced using $\mathsf{RoundBroadcast}$ for each round (either by relying on a period random beacon as in [19] or a committee that will be elected at round ρ as in [15]). This information will be published on the board. For the decryption of ct_b and publishing all unused k values, we piggyback that on the tasks that the board validators do. Thus, the validators keep a record of all ct_b for each round ρ, and when π_ρ becomes available, they decrypt ct_b and post all unused k values on the board. Furthermore, $\mathsf{RoundBroadcast}$ as defined in Definition 1 involves a secret value s that is used to produce the round decryption information. This value (and whether it is needed) is based on the concrete instantiation of the TLE scheme (e.g., using the scheme in [19], s will be shared among the producers of the threshold random beacon).

As noted, during revocation or when the delegation period ends, multiple k values will be published on the board. One may argue that such mass production, or even the existence of ct_b, may violate delegation anonymity. However, this is not the case since: (1) this information does not contain anything about the identity of the proxy signer or which delegation it is tied to, and (2) the original signer will mimic a similar behavior (i.e., periodic publishing of multiple k values and ct_b) for her own signatures as outlined in the Sign algorithm in Fig. 5.

Policy Enforcement. The construction above enforces only a time policy. To restrict the proxy signer to sign messages that satisfy certain policy policy_m, we can adopt two generic approaches from the literature [17,20].

Public Policy. If policy_m is public, we use the warrant approach. The original signer encodes the conditions that message m must satisfy in policy_m, and signs it using a separate signing key (to prevent a proxy signer from using one of the signing tokens to sign any policy she wishes). Any secure signature scheme can be used to sign policy_m; no need to be a proxy scheme. Thus, an original signer will be known using two public keys: the one used for signing the policy and the one used for the proxy signature. The original signer sends the signed policy as part of $\mathsf{degInfo}$ as discussed previously. For Sign, the original signer can pick any policy_m. Both Sign and $\mathsf{DegSign}$ will output policy_m as part of output signature structure σ. Verifying a signature now additionally includes verifying that policy_m is signed by the original signer, and that m satisfies this policy.

Private Policy. For private policy policy_m, we employ a NIZK proof system, so that a signature includes a proof π attesting that m satisfies policy_m and that this private policy is signed by the original signer as above. policy_m is basically encoded as a function f (or a circuit that encodes the required conditions). Also, the public parameters of the system will include any public parameters needed for the NIZK proof system. In terms of signing, σ will be computed over $m \parallel \pi$ to preserve the proof integrity. Verifying a signature will involve verifying π to ensure that the private policy_m is satisfied.

Denial of Service (DoS) Attacks Against Signers. A signature will be directly rejected if its k value is already published on the board. Earlier we state that the k values are either published by the verifiers (after accepting a valid signature), by the original signer (when executing an early revocation), or by the board validators (when a delegation period ends after decrypting ct_b). However, in practice an attacker may perform a DoS attack against the original or proxy signers by intercepting a signature (before being verified) and publishing its k value on the board to invalidate the signature.

We can address this attack as follows: instead of just logging only the k value of a signature, we publish *the whole signature and the hash of the signed message.* Thus, a signature will be rejected if another valid signature (with the same k value) over a *different message* is already on the board. This also means that ct_b will contain valid signatures over random hashes using the tokens instead of just a list of the k values of these tokens. Thus, when we say a k value, we implicitly refer to a valid signature tied to this k value and the component c in the signature (based on construction) is computed over the hash of m rather than m itself. In blockchain applications, our main target, this is not an issue; either way transactions are already published on the chain with their signatures, which will prevent DoS attacks against signers.

Remark 1 (Validity of Accepted Signatures Outside the Delegation Period). A valid signature produced during the delegation period will stay valid outside that period since the verifier can easily check that it is for the same message in

hand and can be verified as many times as desired. A signature over a different message means a reuse of a token and either way will be rejected.

Bulletin Board Synchronization and Off-Chain Processing Issues. In presenting our construction earlier, we assumed that the board is instantly synced. That is, any information sent to the board will appear directly and all parties will see the updated board state instantly. However, this is not the case in practice; propagation delays and other factors may prevent that. So a verifier might be checking an old state that does not contain the updated list of k values/signatures, which allows a proxy signer to use a token several times (with several verifiers) during this period. Furthermore, our scheme is similar to off-chain processing in blockchains. That is, a signature is handed directly to verifiers who rely on the current state of the board during verification. Similar to the concept of double spending in off-chain payments, a malicious proxy signer may reuse a token to generate several valid signatures each of which is handed to a different verifier at the same time. All these verifiers will accept these signatures since none of these signatures is published on the board yet.

We handle both issues by introducing the concept of *delayed signature acceptance*. A verifier will verify the signature as before, then publish it on the board as above, but will not take any action based on this valid signature—which is basically based on the content of the signed message—until later, e.g., a few rounds later. If at that time this verifier finds out that more than one valid signature (over different messages than what she has) using the same k value appeared on the board, they will reject the signature. Note that in blockchain applications, the signature is destined to the board validators, e.g., it is a signature over transactions to be posted on the board. Hence, this issue is resolved by consensus itself. That is, only one valid signature associated with a given k value will be accepted and parties act based on the confirmed state of the blockchain.

Reducing Information Lookup Time. As noted, a verifier has to check that the k value (or basically the signature) is not already published on the board (under $\mathsf{state}[\mathsf{vk}]$) before accepting a valid signature. The cost of information lookup grows linearly with the number of signatures produced under a given vk. This cost can be reduced by using, e.g., universal accumulators [10,26] that are dynamic and allow for non-membership proofs. For each vk, the board validators compute an accumulator of the associated k values and they update it as more k values are published. A signature will be accompanied with a non-membership proof proving that its k value is not in the accumulator. Thus, a verifier just needs to verify the validity of this proof.

5 Security

In the full version [2], we prove the following theorem (where $\mathsf{schnorr\text{-}koe}$ is the Schnorr knowledge of exponent assumption [5,12]).

Theorem 1. *Assuming EUF-CMA security of Schnorr signatures, the* schnorr-koe *assumption, a secure bulletin board, a CCA-secure TLE scheme, an EUF-CMA secure signature scheme, and a secure NIZK proof system,* RelaySchnorr *is an anonymous, timed and revocable proxy signature scheme (cf. Definition 2).*

Acknowledgements. We thank Yolan Romailler for insightful discussions about timelock encryption and drand. The work of G.A. is supported by Protocol Labs grant program RFP-013: Cryptonet network grants, and in part by NSF Grant No. CNS-2226932.

References

1. Afia, I., AlTawy, R.: Traceable policy-based signatures with delegation. In: Deng, J., Kolesnikov, V., Schwarzmann, A.A. (eds.) CANS 2023. LNCS, vol. 14342, pp. 51–72. Springer, Singapore (2023). https://doi.org/10.1007/978-981-99-7563-1_3
2. Almashaqbeh, G., Nitulescu, A.: Anonymous, timed and revocable proxy signatures. Cryptology ePrint Archive (2023). https://eprint.iacr.org/2023/833
3. Ateniese, G., Hohenberger, S.: Proxy re-signatures: new definitions, algorithms, and applications. In: CCS (2005)
4. Backes, M., Meiser, S., Schröder, D.: Delegatable functional signatures. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 357–386. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_14
5. Bellare, M., Crites, E., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022. LNCS, vol. 13510, pp. 517–550. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15985-5_18
6. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_25
7. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998). https://doi.org/10.1007/bfb0054122
8. Bobolz, J., Diaz, J., Kohlweiss, M.: Foundations of anonymous signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Financial Cryptography and Data Security (2024)
9. Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. J. Cryptol. **25**(1), 57–115 (2012)
10. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 561–586. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_20
11. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, p. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: PKC (2014)-519. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_29

12. Crites, E., Komlo, C., Maller, M.: How to prove Schnorr assuming Schnorr: security of multi-and threshold signatures. Cryptology ePrint Archive (2021)
13. Das, M.L., Saxena, A., Gulati, V.P.: An efficient proxy signature scheme with revocation. Informatica **15**(4), 455–464 (2004)
14. Derler, D., Hanser, C., Slamanig, D.: Privacy-enhancing proxy signatures from non-interactive anonymous credentials. In: Atluri, V., Pernul, G. (eds.) DBSec 2014. LNCS, vol. 8566, pp. 49–65. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43936-4_4
15. Döttling, N., Hanzlik, L., Magri, B., Wohnig, S.: McFly: verifiable encryption to the future made practical. In: Baldimtsi, F., Cachin, C. (eds.) FC 2023. LNCS, vol. 13950, pp. 252–269. Springer, Cham. (2023). https://doi.org/10.1007/978-3-031-47754-6_15
16. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: CCS (1998)
17. Fuchsbauer, G., Pointcheval, D.: Anonymous proxy signatures. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 201–217. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85855-3_14
18. Fuchsbauer, G., Pointcheval, D.: Anonymous consecutive delegation of signing rights: unifying group and proxy signatures. In: Cortier, V., Kirchner, C., Okada, M., Sakurada, H. (eds.) Formal to Practical Security. LNCS, vol. 5458, pp. 95–115. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02002-5_6
19. Gailly, N., Melissaris, K., Romailler, Y.: tlock: Practical timelock encryption from threshold BLS. Cryptology ePrint Archive, Paper 2023/189 (2023)
20. Hanser, C., Slamanig, D.: Blank digital signatures. In: ASIA CCS (2013)
21. Hanser, C., Slamanig, D.: Warrant-hiding delegation-by-certificate proxy signature schemes. In: Paul, G., Vaudenay, S. (eds.) INDOCRYPT 2013. LNCS, vol. 8250, pp. 60–77. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03515-4_5
22. Herranz, J., Sáez, G.: Verifiable secret sharing for general access structures, with application to fully distributed proxy signatures. In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 286–302. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45126-6_21
23. Kim, S., Park, S., Won, D.: Proxy signatures, revisited. In: Han, Y., Okamoto, T., Qing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 223–232. Springer, Heidelberg (1997). https://doi.org/10.1007/bfb0028478
24. Lee, B., Kim, H., Kim, K.: Strong proxy signature and its applications. In: Proceedings of SCIS, vol. 2001, pp. 603–608 (2001)
25. Lee, J.Y., Cheon, J.H., Kim, S.: An analysis of proxy signatures: Is a secure channel necessary? In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 68–79. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36563-x_5
26. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 253–269. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72738-5_17
27. Liu, Z., Hu, Y., Zhang, X., Ma, H.: Provably secure multi-proxy signature scheme with revocation in the standard model. Comput. Commun. **34**(3), 494–501 (2011)
28. Lu, E.J.L., Hwang, M.S., Huang, C.J.: A new proxy signature scheme with revocation. Appl. Math. Comput. **161**(3), 799–806 (2005)

29. Malkin, T., Obana, S., Yung, M.: The hierarchy of key evolving signatures and a characterization of proxy signatures. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, p. Malkin, T., Obana, S., Yung, M.: The hierarchy of key evolving signatures and a characterization of proxy signatures. In: EUROCRYPT (2004)-322. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_19
30. Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: CCS (1996)
31. Okamoto, T., Tada, M., Okamoto, E.: Extended proxy signatures for smart cards. In: ISW 1999. LNCS, vol. 1729, pp. 247–258. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-47790-X_21
32. Morita, H., Schuldt, J.C.N., Matsuda, T., Hanaoka, G., Iwata, T.: On the security of the Schnorr signature scheme and DSA against related-key attacks. In: Kwon, S., Yun, A. (eds.) ICISC 2015. LNCS, vol. 9558, pp. 20–35. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30840-1_2
33. Neuman, B.: Proxy-based authorization and accounting for distributed systems. In: International Conference on Distributed Computing Systems (1993)
34. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptol. **13**(3), 361–396 (2000)
35. Schuldt, J.C.N., Matsuura, K., Paterson, K.G.: Proxy signatures secure against proxy key exposure. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 141–161. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78440-1_9
36. Sun, H.M.: Design of time-stamped proxy signatures with traceable receivers. IEE Proc.-Comput. Digit. Tech. **147**(6), 462–466 (2000)
37. Sun, H.M., Lee, N.Y., Hwang, T.: Threshold proxy signatures. IEE Proc.-Comput. Digit. Tech. **146**(5), 259–263 (1999)
38. Varadharajan, V., Allen, P., Black, S.: An analysis of the proxy problem in distributed systems. In: IEEE Computer Society Symposium on Research in Security and Privacy (1991)
39. Wang, H., Pieprzyk, J.: Efficient one-time proxy signatures. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 507–522. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40061-5_32
40. Wuille, P., Nick, J., Ruffing, T.: Schnorr signatures for secp256k1. Bitcoin Improvement Proposal 340 (2020)
41. Xu, S., Yang, G., Mu, Y., Ma, S.: Proxy signature with revocation. In: Liu, J.K., Steinfeld, R. (eds.) ACISP 2016. LNCS, vol. 9723, pp. 21–36. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40367-0_2
42. Zhang, F., Kim, K.: Efficient ID-based blind signature and proxy signature from bilinear pairings. In: Safavi-Naini, R., Seberry, J. (eds.) ACISP 2003. LNCS, vol. 2727, pp. 312–323. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45067-X_27