Anonymous, Timed and Revocable Proxy Signatures

Ghada Almashaqbeh[1]* and Anca Nitulescu[2]

[1] University of Connecticut, ghada@uconn.edu
[2] Protocol Labs, anca.nitulescu@protocol.ai

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 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 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 (which could be of independent interest). Second, we utilize a public bulletin board 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; no trusted party is involved, and 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 introduce several design considerations addressing issues related to deployment in practice.

1 Introduction

Proxy signature schemes are a type of digital signatures that allows one user (the original signer) to delegate their signing right to another party (called proxy signer). The proxy signer can generate a proxy signature that is verified using the original signer's certified public key.[3]

Proxy signatures are useful in many applications where delegation of signing rights is important, such as distributed systems, e-cash using smart cards, grid

*Supported by Protocol Labs grant program RFP-013: Cryptonet network grants.

[3]This is not to be confused with proxy re-signatures [5, 11], 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.

computing, and workload sharing [26, 48, 49, 58]. For example, Alice can let her assistant (Bob) reply to (and sign) her emails while on a vacation, or simply to share the workload of handling emails with Bob. Anonymous delegation guarantees that for the outsider world, everything appears to be done by Alice and 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 [45] by Mambo, Usuda, and Okamoto. In their seminal work, several types of delegation were presented: full delegation—where the proxy signer and the original signer share the same secret key, partial delegation—where the original signer generates a delegation key from its private key that is given to the proxy signer, and delegation by warrant—the original signer issues a policy that restricts the power of the proxy signer with respect to which messages can be signed.

Since then, a large number of followup works emerged on the topic of proxy signatures analyzing security and efficiency of older schemes, and devising new constructions, e.g., [6, 22, 37, 39, 40, 46, 59]. Other foundational works focused on formulating and unifying the security requirements of proxy signatures. For example, Boldyreva et al. [12] proposed a comprehensive security model for the delegation-by-warrant. Malkin et al. [44] formulated a notion for hierarchical proxy signatures, and showed that proxy signatures are equivalent to other signature notions. Schuldt et al. [54] strengthened security by allowing the adversary to query arbitrary proxy signing keys. Furthermore, several works focused on extending proxy signatures to handle different settings and support new features such as: threshold proxy signatures [36, 56], blind proxy signatures [62], and anonymous proxy signatures [28].

Motivation. Our motivation for this work stems from recent advances in distributed systems and Web 3.0 applications, and their need for delegation of signing rights. In particular, we focus on the properties and capabilities that such systems require from delegation. We begin with two motivating applications, outlining these properties, after which we explain how existing solutions are not suitable as there is no single scheme that supports *all* the required properties.

Application 1: DeFi and wallet management. In decentralized finance (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, such that the original account owner retains full control of the credit limit, activation, and deactivation of the new card.) Delegated signatures can easily enable that: Alice delegates signing rights to her sister Eve, under a *policy* ensuring that only transactions with capped values can be issued, and that delegation is valid only for a *given period*, e.g., next two days. Alice can also change her mind and end the delegation earlier if desired. Given the decentralized nature of blockchain-based systems, Alice wants to do all of that in a *decentralized* way without involving any trusted party.

Another usecase is related to cold and hot wallets in cryptocurrencies. Cold wallets are used to store most of the funds and, for security reasons, are not connected to the Internet. It happens that the hot wallet (a mobile app holding a small fund amount, for example) may need more funds than anticipated for some activity. 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 addresses/accounts controlled by the cold wallet, reducing security risks.

Application 2: system robusteness. Another important application is related to system robustness and addressing targeted attacks. Take Byzantine agreement-based consensus as an example. For each round (or epoch) there will be a committee to agree on the next block to be mined. 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 her behalf after a preset timeout. So, even if Alice is a victim of a targeted attack, Bob will do the work until Alice recovers. This improves liveness as it reduces the chances of having empty rounds which, due to lack of enough votes or signatures, will not reach agreement on the next block. Here *anonymity* is a key; outsiders must not know who the backups are to avoid targeted attacks against Bob (and other backups) as part of attacking Alice. The same analogy can be applied to any system in which designated parties must be available to support particular functionalities.

The previous applications outline several desired properties that delegation should achieve. *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. This is particularly important for protecting privacy and handling targeted attacks. Also, delegation should be of an *ephemeral nature*; the proxy signer can exercise the delegated power during a preset time period. Another important feature is *revocability* of proxy signing rights. This could be automatic when the delegation period is over, or on-demand due to unforeseen circumstances, e.g., punishing a misbehaving proxy or that delegation is not needed anymore. Moreover, *policy enforcement* allows the original signer to restrict the proxy signer's power, e.g., only messages belonging to a certain class can be signed.

Furthermore, *decentralization and non-interactivity* are important features especially for large-scale distributed systems. That is, only the original signer and proxy signer are involved; no trusted/semi-trusted entity is needed, and the original signer can send all the delegation information in one shot; no further interaction is needed when the proxy signer exercises the delegated rights. These agree with the spirit of delegation—the original signer can go offline once the delegation information is sent—and they promote scalability.

Despite the large amount of work around signature delegation in the literature, 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 (as we detail in Section 1.2). Full delegation, by giving away the signing key, offers full anonymity but at the price of losing control over the delegation. Many schemes allow fine-grained conditions for the delegation [22, 34, 45], but violate anonymity. Others [6, 28] support controlled and anonymous delegation, but without any revocation capability or timed notion. At the same time, the schemes that support timed delegation and/or revocation [42, 43, 55, 61] either do not offer anonymity, require interaction between the involved parties, 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 and Technical Overview

In this paper, we answer this question in the affirmative by defining a formal notion for anonymous, timed and revocable proxy signature, and constructing a proxy signature scheme satisfying all the properties discussed above. We also discuss challenges that may arise in practice and devise solutions for them. We elaborate on these contributions and the techniques we develop in what follows.

Formal modeling. Our notion builds on previous definitions for proxy signatures [13, 28, 44] and extend them to cover the additional properties that we require. In particular, we generalize the notion of delegation to produce a generic delegation information rather than restricting these to be tokens or delegation keys. We also introduce a new algorithm for revocation that covers two revocability flavors: automatic that ends a delegation when the preset time period is over, and on-demand to enable the original signer to end a delegation earlier than envisioned. For policy enforcement, we view the policy as two part: a policy over time encapsulating the delegation period, and a policy over the message restricting the class of messages that can be signed. Any, or even both, of these policies can be empty allowing for unrestricted delegation on any, or both, of these dimensions.

Our notion includes only one verification algorithm that is used for all signatures regardless of whether they are generated by the original or proxy signer. Verification is done with respect to the original signer's public key and does not involve the proxy signers' identities. This ensures that signatures are indistinguishable. Our verification algorithm checks compliance with the policy and that the signature is not revoked. We formally define correctness and security of proxy signatures, where the latter covers unforgeability under chosen message attacks, anonymity, revocability, and policy enforcement. We note that none of the previous definitions in the literature covered all these properties at once, and none of them defined a time policy or revocation.

Construction. We introduce a new proxy signature scheme that realizes our notion. Our construction combines Schnorr signature, timelock encryption, and a public bulletin board to support token-based delegation that is anonymous and revocable, and allow time and message policy enforcement. In detail, 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. The token is a Schnorr signature (over a random value k) using the original signer's signing key, thus, only the original signer can produce tokens. The token is then used as a signing key for another invocation of Schnorr in order to sign the intended message. Thus, for delegation, the original signer can produce tokens (on her own without interacting with the proxy) and communicate them securely to the proxy who can use them to sign messages. Signature verification, naturally, includes two checks, instead of one as in original Schnorr signatures, to verify correctness of both layers. This 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 signature size, a signature consists of four field elements and one group element. This means the cost of delegation consists of one group element and two field elements compared to the size of original Schnorr signatures.

We enforce *one-time use* of a token by publishing the unique value associated to a valid signature (namely, k mentioned above) on the bulletin board. The verifier first checks that the k value of the signature is not already on the board, and if so, the verifier rejects the signature. After verifying a valid signature, the verifier publishes the new k value on the board. Thus, a proxy signer cannot reuse a token to sign several messages. This is different from the conventional notion of one-time signatures, where if a signer signs more than one message, her signing key will be revealed. Since we work in the delegation model, 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 more than one message, and still only one of these signatures will be accepted by the verifier.

For the *timed delegation*, we also utilize the public bulletin board along with the recent notion of timelock encryption [30]. In timelock encryption, a ciphertext is generated with respect to time ρ. When that time comes, some public information will become available allowing for decrypting that ciphertext. In our scheme, to enforce a time period $[\rho_a, \rho_b]$, meaning that a proxy signer cannot exercise the delegation outside this period, the original signer encrypts the delegation information (composed of u signing tokens) locked to time ρ_a. She sends this ciphertext privately to the proxy signer. Only when time ρ_a comes, the proxy signer can access these tokens. To enforce the end of the period, which is basically the *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 these and publish all unused k values preventing the proxy signer from using any unused tokens after time ρ_b.

For *on-demand revocation*, we do that in a similar way to the automatic revocation. The difference is that the original signer publishes the unused k values of the delegated tokens on the board. The original signer can do that at anytime before time ρ_b, or even before time ρ_a cancelling the whole delegation.

So even if the proxy signer tries to use these tokens to produce signatures, these signatures will be rejected during verification. We note that existing schemes that tried to support revocability, and the timed notion, require a trusted third party, thus introducing trust issues and compromising anonymity. To the best of our knowledge, we are the first to support these properties in an anonymous and fully decentralized way.[4]

For *policy enforcement over messages*, we follow generic approaches from the literature [28, 34] for two cases: public policy and private one. *Public policy* is enforced using 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 from the one used in the delegation. Verifying a signature in this case involves verifying that policy_m is signed by the original signer and that m satisfies policy_m. *Private policy* is enforced using a non-interactive zero knowledge proof (NIZK) system. A signature in this case will include a proof π attesting that the signed message satisfies a private policy policy_m, and that policy_m is signed by the original signer. Thus, verifying a signature involves verifying the proof π.

Security. We formally prove security of our construction based on our notion for anonymous, timed and revocable proxy signatures. Unfrogeability relies on the unforgeability of Schnorr signatures in the random oracle model [51], and the Schnorr knowledge of exponent assumption in the algebraic group model [20]. 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, and that verification is identical for both and does not depend on the identity of the proxy signer. The one-time property relies on the security of the public bulletin board (by being an append-only, publicly accessible log maintained by validators with honest majority). Revocability (both automatic and on-demand) relies on the correctness and CCA security of timelock encryption as well as the security of the board. Lastly, policy enforcement relies on the security of digital signatures (for public warrants) or security of NIZKs (for private policies), CCA security of the timelock encryption scheme, and the security of the board.

Challenges. A few challenges arise when considering the use of our scheme in practice. These mainly stem from relying on the bulletin board to enforce the time notion and the one-time property. Examples include synchronization issues of the board and whether delays in publishing the unique values of the signatures may allow a malicious proxy signer to reuse a token several times instead of once. Another is related to denial of service attacks against the signers, where an attacker may intercept a signature and publish its k value before being verified, thus invalidating a legitimate signature. This is in addition to anonymity

[4]Note that we assume the bulletin board to be an append-only, publicly-accessible log that is generally instantiated as a blockchain maintained by a set of miners or validators rather than a single trusted party. This is done to achieve decentralization in the same spirit as in blockchain-based systems. The time ρ for us in this case is a round number from the blockchain, i.e., a block with a given index.

concerns related to mass publication of k values during revocation, which may reveal that a delegation took place. We discuss these and other challenges, along with solutions to handle them in Section 5.

Threshold delegation. An interesting direction is to delegate signing rights to a committee rather than a single party [36], and require that at least t parties participate in signing a message. This trust distribution is valuable in many applications, e.g., requiring several hot wallets (owned by the same user) to sign in order to get funds out of a cold wallet. We briefly discuss how to realize this as an extension to our proxy signature scheme, based on prior works on threshold key distribution and threshold Schnorr signature.

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. Same for our tokenized version of Schnorr signatures; it also could be of independent interest allowing for anonymous delegation without requiring a bulletin board (less restricted delegation that does not involve time or revocation). A proxy signer (or even the original signer herself) can use a single token to sign any number of messages without compromising security (i.e., without revealing the original signers' signing key or the token). 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 [60]).

1.2 Related Work

We review existing proxy signature schemes in terms of the properties they support (based on the list of properties that we aim to achieve), showing that none of them support all these properties. Then we review works on relevant notions, including one-time signatures and timed cryptographic primitives, and discuss how our work is positioned with respect to these efforts.

1.2.1 Proxy signatures

Anonymity. Anonymity is usually not supported since the proxy signer's key is public and needed in the verification process, e.g., [45]. Fuchsbauer et al. [28, 29] address this issue by unifying the notion of proxy and group signatures, and they consider traceability where a trusted authority holding a trapdoor can compromise anonymity if needed. At a high level, the signature is basically a NIZK attesting that the signer belongs to a group (recognized by the original signer) without revealing the group public keys. Beside introducing a centralized entity, this scheme does not support revocation or timed delegation. Functional signatures [15] 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). Similarly, the signature is a NIZK attesting that the proxy has sk_f and that m satisfies f. Delegatable functional signatures [6] utilize signature malleability to allow delegation (the original signer generates a signature over a message m that a proxy signer can maul to become a signature over m' that satisfies f). Both notions support delegation anonymity (all signatures are verified against the original signer's key) but not timed or revocable delegation. Our anonymous proxy signature scheme supports timed and revocable delegation.[5]

Time-bounded delegation. The few works that discussed a timed notion for delegation are limited. Lu et al. [43] add the delegation period to the warrant, and relies on a trusted server to enforce this period by issuing a timestamp for each signature a proxy signer wants to generate. Sun et al. [55] 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 having a trusted server and the interactivity requirement, these schemes do not support anonymity and do not have formal security analysis. We support timed delegation in a decentralized and non-interactive way, through the use of a public bulletin board and timelock encryption.

Revocation. Techniques for revoking delegation rights [45, 61] 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 [61] use revocation periods (or epochs), where for each signature, the proxy signer generates a proof that it is not on the epoch revocation list (proxy signers' public keys are known). These approaches compromise anonymity. Other schemes [21, 42, 43] rely on a (semi-)trusted server to enforce revocation. 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 and without requiring the original signer to change her signing key.

Policy enforcement. Warrants are used to enforce a policy over delegation, and are usually public—so a verifier rejects any signature over a message that does not obey the warrant [13, 44]. Supporting private warrants mainly relies on non-interactive zero knowledge proofs (NIZKs) to show that a signed message belongs to a hidden (committed) set of messages [35], on polynomial commitments to restrict the proxy signer to sign messages following a specific template [34], or on anonymous non-interactive credentials [22]. However, non of these scheme offer proxy signer's anonymity. Functional signatures and delegatable functional signatures [6, 15], mentioned earlier, support private policy and proxy anonymity using NIZKs to prove that message m satisfies the policy encoded as function f. In terms of policy enforcement over the signed message, our work use these generic approaches, namely public warrants or private ones via NIZKs, but it adds the support of anonymous, timed and revocable delegation as mentioned above.

[5] As for accountability or traceability, we leave it as a future work since we want to avoid relying on a trusted entity to enforce that.

1.2.2 One-time Signatures

One-time signatures are schemes that allow signing only one message using a given signing key [2, 24, 38]. The one-time notion is enforced by the fact that if more than one signature is produced, these signatures can be used to reveal the signing secret key. Such a notion does not work in the anonymous delegation setting: the signing keys are tied to their own public keys (that are used in verification) rather than to the original signer's public key. Even if there is a way to produce tokens tied to the original signer public key, under this one-time security guarantee, a proxy signer could reuse a token in order to reveal the original signer's signing key.

One-time signatures have also been investigated in the quantum model [3, 10, 19] and permit signature tokenization; the original signer generates a signing token that a proxy signer can use to sign a message m. The one-time is achieved by generating an unclonable (quantum state) token that self-destructs once it is used. Our goal is to support delegatable one-time signing tokens in the classical model. Furthermore, none of these schemes support timed delegation; a proxy signer can use the token at anytime. While some of these works discuss a limited notion of revocability that is interactive; the original signer notifies the proxy signer that she wants to revoke, and the proxy signer has to produce a proof that the token has been used or simply return the physical token. Our revocation notion does not require any interaction with the proxy signer.

1.2.3 Timed Cryptographic Primitives

There are many versions of conventional cryptographic primitives that support a time notion (we briefly review some of them here). These started with time capsules and timed commitments [7, 14, 52], which can be opened after performing a sequential computation for a preset time period. More recent notions include, for example, short lived proofs and signatures [4] that support deniability. That is, a proof or signature is valid for a time period—the time needed to open the capsule—after which it can be forged. Timed signatures [31, 57] allow locking a signature on a known message for a given amount of time through means of times capsules as well. Time-release encryption allows encrypting messages to the future such that before that time even the designated recipient cannot decrypt. Generally, existing constructions either relied on time capsules, e.g. [52], or involved a trusted server, e.g. [17].

Timelock encryption (TLE) aims to avoid the high cost of opening the capsules and the server trust assumption. TLE constructions rely on a reference time realized by a blockchain. Liu et al. [41] use a blockchain along with witness encryption to build a TLE scheme. In encryption to the future [16] similar tools are used to encrypt messages that can be decrypted by a future committee to be elected. McFLY [23] uses signature-based witness encryption combined with a BFT (Byzantine fault tolerant) blockchain, so the decryption of the message is piggypacked on the tasks a blockchain committee (elected at a certain time) is doing. Another timelock encryption scheme is proposed in [30] that relies on a (threshold) random beacon to produce decryption information at certain time allowing anyone to decrypt. Lastly, Abadi et al. [1] uses publicly verifiable random beacons to realize the notion of timestamping for signatures and zero-knowledge proofs. Our work extends these efforts to involve a timed notion for delegation using means of TLE that also enables us to support decentralized revocation.

2 Preliminaries

2.1 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 denoted lowercase. Elements in a multiplicative group $\mathbb{G}$ of order q generated by generator $G \in \mathbb{G}$ are denoted by capital letters. We let $\lambda \in \mathbb{N}$ denote the security parameter, pp denote the public parameters, and PPT be a shorthand for probabilistic polynomial time.

We say that a function is *negligible* in λ, and we denote it by $\mathsf{negl}(\lambda)$, if it is a $\mathcal{O}(\lambda^{-c})$ for any fixed constant c. We also say that a probability is *overwhelming* in 1^λ if it is $1 - \mathsf{negl}(\lambda)$.

2.2 The Schnorr Digital Signature Scheme

We recall the Schnorr digital signature scheme, which we rely on in our construction as mentioned previously. Schnorr signature is obtained by applying the Fiat-Shamir transform [25] to the Schnorr identification scheme [53]. We adopt the formulation from [47] that mitigates related key attacks.

Related key attacks. In related key attacks (RKA), formalized by Bellare and Kohno [8], an adversary may alter a (hardware-stored) secret key and obtain signatures under the modified key. This notion captures security against practical attacks such as tampering or fault injection. Morita et al. [47] showed that the original version of Schnorr signature is vulnerable to related key attacks, and proposed an easy fix: include the public verification key in the challenge c component as shown below.

For a security parameter λ, let $\mathbb{G}$ be a cyclic group of a prime order q and a generator G, and $\mathsf{H} : \{0,1\}^* \times \mathbb{G}^2 \to \mathbb{Z}_q$ be a hash function. The Schnorr signature scheme is a tuple of three algorithms $\Sigma_{\mathsf{Schnorr}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ defined as follows:

- $\mathsf{Schnorr.KeyGen}(1^\lambda)$: On input the security parameter λ, choose uniform $x \in \mathbb{Z}_q$ and compute $X = G^x$. Set the secret signing key $\mathsf{sk} = x$ and the public verification key $\mathsf{vk} = X$.
- $\mathsf{Schnorr.Sign}(\mathsf{sk}, m)$: On input the secret key $\mathsf{sk} = x$ and the message m, choose uniform $k \in \mathbb{Z}_q$. Compute $K = G^k$, $X = G^x$, $c = \mathsf{H}(m, X, K)$, and $s = k + cx \mod q$. Output the signature $\sigma = (c, s)$.
- $\mathsf{Schnorr.Verify}(\mathsf{vk}, m, \sigma)$: On input the public key $\mathsf{vk} = X$, the message m, and signature $\sigma = (c, s)$ over m, compute $K = G^s \cdot X^{-c}$ and $c' = \mathsf{H}(m, X, K)$, then output 1 if $c = c'$.

Correctness and security. It is easy to see that for any correctly generated signature, Verify will always output 1. Existential unforgeability against adaptive chosen message attacks (EUF-CMA) of Schnorr signature in the random oracle model has been formally proved in [51] under the Discrete Logarithm assumption by relying on the forking lemma.

2.3 Timelock Encryption

Timelock Encryption (TLE) is a cryptographic primitive in which messages can be encrypted towards a time ρ such that they can only be decrypted after that time. This time ρ is basically a round number (that could be realized using, e.g., a blockchain). A time-related information π_ρ is published to enable decryption at each time ρ. We are interested in a publicly decryptable variant of TLE, where ciphertexts can be decrypted by any party given only π_ρ. We adopt the definition in [30] while using a more generalized time information production algorithm (i.e., $\mathsf{RoundBroadcast}$) as shown below.[6]

Definition 1 (Timelock Encryption (TLE)). *A Timelock Encryption scheme $\mathcal{E}$ is a tuple of five polynomial-time algorithms:*

$\mathsf{TLE.Setup}(1^\lambda) \to (\mathsf{pp}, s)$**:** *a probabilistic algorithm that takes as input the security parameter λ, and outputs public parameters pp and a private key s.*[7]

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

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

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

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

Correctness requires that for all λ, all $(\mathsf{pp}, s) \leftarrow \mathsf{Setup}(1^\lambda)$, all ρ and all m, if $(\mathsf{ct}_\rho, \tau) \leftarrow \mathsf{Enc}(\rho, m)$ and $\pi_\rho \leftarrow \mathsf{RoundBroadcast}(s, \rho)$, then:

$$\mathsf{Dec}\big(\rho, \pi_\rho, \mathsf{ct}_\rho\big) = \mathsf{PreOpen}\big(\mathsf{ct}_\rho, \tau\big) = m$$

For security, various security models have been proposed and analyzed in [18]. We require a TLE scheme to be secure against a CCA attacker (given that anyone can decrypt once the decryption information becomes available, this implies that the attacker has access to the decryption oracle).

[6]The definition in [30] capture a threshold-based algorithm definition where multiple parties generate the timing information. We keep the definition general and leave any such details to the instantiation.

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

In our proxy signature scheme, 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 (we require this instantiation to be fully decentralized and publicly verifiable). Gailly et al. [30] develop a CCA-secure TLE scheme that realizes the notion above with these requirements. Furthermore, McFly [23] in a way can be used to realize this notion as well: the committee elected at round ρ will decrypt the ciphertext making it available to everyone, and the original signer who knows the plaintext in our setup can execute the $\mathsf{PreOpen}$ by simply revealing that plaintext.

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 [13,28,44] and extend them to cover the additional properties we require.

Definition 2. *An anonymous, timed and revocable proxy signature scheme* $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Delegate}, \mathsf{DegSign}, \mathsf{Revoke}, \mathsf{Verify})$ *is a tuple of seven* PPT *algorithms defined as follows:*

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$: *Takes the security parameter* λ *as input, and outputs a set of public parameters* pp.[8]

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

$\mathsf{Sign}(\mathsf{sk}, m, \mathsf{policy}) \rightarrow (\sigma, \theta)$: *Takes the signing key* sk*, a message* m*, and a policy* policy *as inputs, and outputs a signature* (σ, θ) *over* m *(where* σ *is the part computed over* m *and* θ *contains information needed to verify* σ*).*[9]

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

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

$\mathsf{Revoke}(\mathsf{degInfo}, \mathsf{rk}) \rightarrow \mathsf{revState}$: *Takes the delegation information* $\mathsf{degInfo}$ *and revocation key* rk *as inputs, and outputs revocation state* $\mathsf{revState}$.

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

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

[9]Having these two parts allows us to formulate our delegation-related definitions in a cleaner way since part of θ comes from the delegation, while σ is produced when the message is ready to be signed.

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, beside the message to be signed and the secret signing key, Sign also takes a policy as input. This specifies any conditions that the produced signature must be compliant with in order to be accepted. We view a policy to be mainly composed of two sub-policies $(\mathsf{policy}_m, \mathsf{policy}_t)$: policy_m determines which messages can be signed—so a signature over $m \notin \mathsf{policy}_m$ will be rejected, while policy_t determines the time period—so a signature produced and verified outside this period will be rejected. Although the latter is primarily needed for delegation (to enforce the timed notion) we include it here to satisfy anonymity. That is, all signatures whether produced by the original or proxy signer will have the same structure. Our definition is general in the sense that the term policy is a generic one that could include additional polices, other than policy_m and policy_t, if desired.

For delegation, $\mathsf{Delegate}$ takes as input the delegation specifications $\mathsf{degspec}$ that include any auxiliary information needed to process the delegation, and any policy the original signer wants to enforce over the delegation (again, we focus on policy_m and policy_t in this work). $\mathsf{Delegate}$ produces the delegation information $\mathsf{degInfo}$ that contains all information a proxy signer needs to sign on behalf of the original signer, e.g., keys or signings tokens, and the policies to be enforced. Thus, both Sign and $\mathsf{DegSign}$ (the latter is used by the proxy signer to sign) take the policy as input. The original signer can produce any policy she wishes, where the empty policy $\mathsf{policy} = \perp$ means that any message can be signed and signatures can be produced/verified at anytime.

The signature produced by Sign and $\mathsf{DegSign}$ is composed of two fields σ and θ. These are tied to each other where σ is computed over the message m, and θ includes all information needed to verify σ and to ensure that the signature is compliant with the specified policy. As shown, our definition contains one verify algorithm that is used to verify any signature, whether produced by the original or proxy signer. This is done to support anonymity: both signatures will have same format (so they are indistinguishable) and both are verified against the same verification key of the original signer. Verify will reject any signature that does not satisfy the time and message policies. Furthermore, Verify takes the revocation state $\mathsf{revState}$ as input, and thus it checks that the signature is not revoked (and so if revoked, the signature will be rejected). Combined with the Revoke algorithm, our definition captures the capability of revoking a delegation (using the revocation key rk produced by $\mathsf{Delegate}$).

Now we define the properties listed in Definition 2. We use code-based games (or experiments) [9] 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. We first introduce some intuition for each security property, and then give a formal definition.

Correctness. Informally, correctness implies that a signer holding a valid secret key or delegation information can always produce a valid signature (σ, θ) over a

```
$\mathsf{Exp}^{\mathsf{ProxyEUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda)$
1 :  $\mathsf{L_{sign}} \leftarrow \varnothing, \mathsf{L_{deleg}} \leftarrow \varnothing$
2 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
3 :  $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
4 :  $O \leftarrow \{O\mathsf{Sign}, O\mathsf{Delegate}\}$
5 :  $(m^*, \sigma^*, \theta^*) \leftarrow \mathcal{A}^O(\mathsf{vk})$
6 :  if $m^* \in \mathsf{L_{sign}} \lor \theta^* \in \mathsf{L_{deleg}}$
7 :    return $0$
8 :  if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*, \theta^*, \perp) = 0$
9 :    return $0$
10 : return $1$

$O\mathsf{Sign}(m, \mathsf{policy})$
1 :  $(\sigma, \theta) \leftarrow \mathsf{Sign}(\mathsf{sk}, m, \mathsf{policy})$
2 :  $\mathsf{L_{sign}} \leftarrow \mathsf{L_{sign}} \cup \{m\}$
3 :  return $(\sigma, \theta)$

$O\mathsf{Delegate}(\mathsf{vk}, \mathsf{degspec})$
1 :  $(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
2 :  $\mathsf{L_{deleg}} \leftarrow \mathsf{L_{deleg}} \cup \{\mathsf{degInfo}\}$
3 :  return $(\mathsf{degInfo}, \mathsf{rk})$
```

Fig. 1: Existential unforgeability under chosen message attacks.

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 revState; and that it does not violate the specified policy.

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

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

Existential unforgeability. This property states that no adversary can produce a valid signature without the knowledge of at least one of the following: the signing key sk or delegation information degInfo created with respect to (sk, 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 Figure 1, and the probability is taken over all randomness used in the experiment.

We note the following in the description of $\mathsf{Exp}^{\mathsf{ProxyEUF\text{-}CMA}}_{\Sigma,\mathcal{A}}$. First, checking if the forged signature has been produced by a valid delegation obtained through $O\mathsf{Delegate}$ is done by checking if θ^* is in the list of delegation queries $\mathsf{L_{deleg}}$. That

```
$\mathsf{Exp}_{\Sigma,\mathcal{A}}^{\mathsf{DegAnon}}(\lambda)$
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 :  $(\bar{\sigma}, \bar{\theta}) \leftarrow \mathsf{Chal}_b(m^*, \mathsf{degspec})$
7 :  $b^* \leftarrow \mathcal{A}^O(\bar{\sigma}, \bar{\theta})$
8 :  if $b^* = b$
9 :    return $1$
10 : return $0$

$O\mathsf{Sign}(m, \mathsf{policy})$
1 :  $(\sigma, \theta) \leftarrow \mathsf{Sign}(\mathsf{sk}, m, \mathsf{policy})$
2 :  return $(\sigma, \theta)$

$O\mathsf{Delegate}(\mathsf{vk}, \mathsf{degspec})$
1 :  $(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
2 :  return $(\mathsf{degInfo}, \mathsf{rk})$

$\mathsf{Chal}_b(m^*, \mathsf{degspec})$
1 :  if $\mathrm{b} = 0$
2 :    Extract $\mathsf{policy}$ from $\mathsf{degspec}$
3 :    $(\sigma_0, \theta_0) \leftarrow \mathsf{Sign}(\mathsf{sk}, m^*, \mathsf{policy})$
4 :  if $\mathrm{b} = 1$
5 :    $(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
6 :    $(\sigma_1, \theta_1) \leftarrow \mathsf{DegSign}(m^*, \mathsf{degInfo})$
7 :  return $(\sigma_b, \theta_b)$
```

Fig. 2: Anonymity for delegation.

is, degInfo will contain (perhaps part of) θ needed to verify a corresponding σ produced using degInfo.[10] Second, the revocation state revState is empty when verifying the forged signature. We define the security notion of the revocation property separately, thus we do not include it here.

Anonymity. This implies that the verifier, or any adversary, will not be able to infer any information about a delegation that took place (one that he does not know its degInfo). In other words, 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 proxy signers, i.e., delegations, in the first place. Thus, all signatures are indistinguishable and all are verified against the original signer's verification key vk. Given that the original signer is the one who produces delegation in a non-interactive way, only her knows that Delegate was invoked. Also, the produced degInfo is transmitted to the proxy signer over a secure channel since it is secret information. Thus, outside these two parties, no one will be able to tell that such information was produced or transferred. Thus, by having indistinguishable signatures that are verified in an identical way, delegation anonymity is satisfied.

[10]Note that this does not mean that θ can identify a delegation, thus compromising anonymity. It just means that parts of the information required for verifying signatures produced using a delegation are generated by the original signer.

```
$\mathsf{Exp}^{\mathsf{DegRev}}_{\Sigma,\mathcal{A}}(\lambda)$
1 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2 :  $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
3 :  $O \leftarrow \{O\mathsf{Sign}, O\mathsf{Delegate}\}$
4 :  $\mathsf{degspec} \leftarrow \mathcal{A}^O(\mathsf{vk})$
5 :  $(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
6 :  $\mathsf{revState} \leftarrow \mathsf{Revoke}(\mathsf{degInfo}, \mathsf{rk})$
7 :  $(m^*, \sigma^*, \theta^*) \leftarrow \mathcal{A}^O(\mathsf{vk}, \mathsf{degInfo})$
8 :  if $\theta^* \notin \mathsf{degInfo}$
9 :    return 0
10 : if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*, \theta^*, \mathsf{revState}) = 0$
11 :   return 0
12 : return 1
```

Fig. 3: Delegation revocation ($O\mathsf{Sign}$ and $O\mathsf{Delegate}$ are as defined in Figure 2).

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 Figure 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 denoted as 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. Informally, 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 Figure 3, and the probability is taken over all randomness used in the experiment.

As shown in the figure, $\mathcal{A}$ chooses the delegation specifications for the delegation that will be created. This delegation is then revoked, and $\mathcal{A}$ is challenged to produce a valid signature (that will be accepted) using this revoked delegation.

```
Exp $^{\mathsf{DegPolicy}}_{\Sigma,\mathcal{A}}(\lambda)$
1 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2 :  $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
3 :  $O \leftarrow \{O\mathsf{Sign}, O\mathsf{Delegate}\}$
4 :  $\mathsf{degspec} \leftarrow \mathcal{A}^O(\mathsf{vk})$
5 :  $(\mathsf{degInfo}, \mathsf{rk}) \leftarrow \mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$
6 :  $(m^*, \sigma^*, \theta^*) \leftarrow \mathcal{A}^O(\mathsf{vk}, \mathsf{degInfo})$
7 :  if $\theta^* \notin \mathsf{degInfo}$
8 :    return 0
9 :  Extract $\mathsf{policy} = (\mathsf{policy}_m, \mathsf{policy}_t)$ from $\mathsf{degspec}$
10 : if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*, \theta^*, \perp) = 1 \wedge$
     $(m^* \notin \mathsf{policy}_m \vee \mathsf{now} \notin \mathsf{policy}_t)$
11 :   return 1
12 : return 0
```

Fig. 4: Delegation policy enforcement (OSign and ODelegate are as defined in Figure 2).

Thus, the game checks that indeed the signature (σ^*, θ^*) returned by $\mathcal{A}$ is produced using the same revoked degInfo. This is done by checking that $\theta^* \in \mathsf{degInfo}$ (where as noted earlier, the verification information θ—or part of it—is included in degInfo).

Policy enforcement. Informally, this implies that an adversary holding a valid delegation (based on specifications of her choice degspec including a policy policy) 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 period.

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 Figure 4, and the probability is taken over all randomness used in the experiment.

As shown in the figure, the experiment checks that the signature returned by the adversary is produced using the delegation created based on degspec (using the same technique in Figure 3). This is done to rule out the trivial attack in which $\mathcal{A}$ submits policy and then creates a valid signature compliant with a different $\mathsf{policy}' \neq \mathsf{policy}$ but not compliant with policy (and hence, wins the game). 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.[11]

4 Construction

We present a construction for 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 that they can use to sign messages. Towards that, we introduce a modified version of Schnorr signature that is tokenizable, and employ a public bulletin board and timelock encryption to enforce the one-time use of signing tokens as well as the timed and revocable properties for delegation.

The full construction is shown in Figures 5 and 6. To simplify the discussion, we present our scheme with only the time policy, as shown in these figures, and we defer enforcing a policy over the messages until the end. We organize the discussion in the section based on the features our scheme supports.

One-time tokenizable signatures. We introduce a one-time tokenizable digital signature scheme based on Schnorr signatures. This is done via a two-layered approach: Layer 1 produces a token which is a Schnorr signature over a fresh random value, while layer 2 uses the token to produce a signature over the intended message. Thus, verifying a signature involves verifying the validity of both signature layers.

For the signing algorithm, we first generate some "token" value z, using the signing key $\mathsf{sk} = x$, that is actually a Schnorr signature on 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 in the delegation. To sign a message m, the original signer uses z as a secret key and produces another Schnorr signature over m with randomness e as shown in the figure.[12] So this signature will be over the value $c = \mathsf{H}(m, Z, E)$, where $Z = G^z$. The output signature is $(\sigma = (w, c, s), \theta = (k, Z))$.

The verification algorithm (in Figure 6) use the public key $\mathsf{vk} = X$ to verify a signature with the format above. In a way, it must verify the two layers of the Schnorr signature applied to the message. This is done by computing the randomness R and E and then verify that the signed hashes, namely, w and c, found in the signature are indeed correct hashes based on the computed randomness.

The one-time property is enforced in the verify algorithm as follows. Recall that the value k is picked fresh for each new signature (and subsequently for each signing token), meaning that it is a unique value tied to the signature. When accepting a valid signature, the verifier will post k on the bulletin board, and only valid signatures with fresh k values that do not appear on the board will

[11] As mentioned earlier, if more policies are needed, other than policy_m and policy_t, these can be added to the definition. In this work we focus on these two policies.

[12] Note that the value z is also a random element in $\mathbb{Z}_p$.

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 $\mathsf{H} : \{0,1\}^* \times \mathbb{G}^2 \to \mathbb{Z}_q$ to be a hash function, initialize $\mathsf{state} = \{\}$, and invoke $\mathsf{TLE.Setup}(1^\lambda)$ to obtain the public parameters pp.

$\mathsf{KeyGen}(1^\lambda)$: 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. Set $\sigma = (w, c, s),\ \theta = (k, Z)$
5. Output the signature $(\sigma, \theta)$
```

Every now and then, S either (1) populates a set klist from the stored k values and fresh values, encrypts it as $(\mathsf{ct}_b, \tau_b) = \mathsf{TLE.Enc}(\mathsf{klist}, \rho_b)$, where ρ_b is some future round number, and then posts ct_b on the board. Or (2) generates some fresh klist and posts it on the board.

$\mathsf{Delegate}(\mathsf{sk}, \mathsf{vk}, \mathsf{degspec})$: 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: $(\mathsf{ct}_a, \tau_a) = \mathsf{TLE.Enc}(t_1 \parallel \cdots \parallel t_u, \rho_a)$ and $(\mathsf{ct}_b, \tau_b) =$
   $\mathsf{TLE.Enc}(\mathsf{klist}, \rho_b)$
4. Set $\mathsf{degInfo} = (\rho_a, \rho_b, \mathsf{ct}_a)$
5. Output ($\mathsf{degInfo}$, $\mathsf{ct}_b \parallel \tau_b$)
```

S stores the ciphertext ct_b and the trapdoor τ_b to be used for revocation if needed (τ_a is simply dropped as it is not needed), posts (ρ_b, ct_b) on the board, and sends $\mathsf{degInfo}$ to P.

Fig. 5: A construction for anonymous, timed and revocable proxy signatures—continued in Figure 6.

be accepted. Consequently, a verifier must first check whether k is on the board state state, and if yes, the signature will be rejected. Looking ahead, this will allow detecting a proxy signer that tries to use the same token to sign several messages.

```
$\mathsf{DegSign}(m, \mathsf{degInfo})$: On input a message $m$ and delegation information $\mathsf{degInfo}$, $P$
does the following (let $\rho_{now}$ 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 $\pi_{\rho_a}$ from the board and set $\mathsf{degInfo} =$
         $(\rho_a, \rho_b, \mathsf{TLE.Dec}(\rho_a, \pi_{\rho_a}, \mathsf{ct}_a))$
     (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) Set $\sigma = (w, c, s)$, $\theta = (k, Z)$ and output $(\sigma, \theta)$

$\mathsf{Revoke}(\mathsf{degInfo}, \mathsf{rk})$: On input $\mathsf{degInfo} = (\rho_b, \mathsf{ct}_b)$ and revocation key $\mathsf{rk}$, do (let $\rho_{now}$
be the current round number):
  1. If $\rho_{now} \geq \rho_b$, then retrieve $\pi_{\rho_b}$ from the board and compute $\mathsf{klist} =$
     $\mathsf{TLE.Dec}(\rho_b, \pi_{\rho_b}, \mathsf{ct}_b)$
  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 $k \in \mathsf{klist} \wedge k \notin \mathsf{state}$ to the board state $\mathsf{state}$

$\mathsf{Verify}(\mathsf{vk}, m, \sigma = (w, c, s), \theta = (k, Z), \mathsf{revState} = \mathsf{state})$: On input the verification
key $\mathsf{vk} = X$, the message $m$, signature $(\sigma = (w, c, s), \theta = (k, Z))$ over $m$, and the
revocation information recorded on $\mathsf{state}$, if $k \in \mathsf{state}$, then output 0. Else, add $k$
to $\mathsf{state}$ 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: A construction for anonymous, timed and revocable proxy signatures (cont.).

To preserve delegation anonymity, every now and then the original signer will post k values (either these that he used when signing or freshly chosen values) on the board. This also can take two forms: publishing a ciphertext (locked to a future time) of these values on the board, or publish a fresh list directly. This is needed to mimic the behavior of signatures produced by delegation as we will see shortly, and thus, satisfy the indistinguishability of signatures.

Timed delegation. To delegate signing, and as shown under the Delegate algorithm in Figure 5, the original signer picks the delegation specification including the number of signing tokens to be generated u and the delegation period. The original signer uses her signing key $\mathsf{sk} = x$ to generate u fresh signing tokens, denoted as $t_1, \ldots, t_u$, for the proxy signer. Each of these tokens contains z (first layer Schnorr signature over k), and the corresponding w and k values.

Our goal is to enforce a time period over the delegation without the help of any trusted/semi-trusted party. This means that the proxy signer cannot use the signing tokens outside the time period specified by the original signer. Here we utilize a recent notion of timelock encryption TLE (defined in Section 2) in the blockchain model (where the bulletin board can be instantiated 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 automatically force a proxy signer to use the signing tokens only during $[\rho_a, \rho_b]$, we propose the following. The original signer uses the TLE scheme to encrypt the tokens $t_1 \parallel \cdots \parallel t_u$ and produce ct_a (with a secret trapdoor τ_a that will not be used) such that when ρ_a comes, and so the decryption information π_{ρ_a} becomes available, the proxy signer can decrypt ct_a to retrieve the tokens. This enforces the beginning of the time period. To enforce the end of the period, recall that any signature with a k value that appears in state (the board state) will be rejected. Thus, we let the original signer use the TLE scheme to produce another ciphertext for time ρ_b, denoted as ct_b, encrypting the list of k values of the tokens, denoted as klist, and post ct_b on the board. When time ρ_b comes, and so π_{ρ_b} becomes available, the board validators will be able to decrypt ct_b and publish all unused k values in klist on the board (this is included under Revoke in Figure 6). This will prevent the proxy signer from using any of the unused tokens after time ρ_b, and thus enforcing the end of the delegation period.

The original signer stores ct_b (and any additional information he might need to identify the delegation associated with ct_b) and the secret trapdoor of ct_b, denoted as τ_b, to 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 Figure 6, once time ρ_a comes, the proxy signer will be able to decrypt ct_a, by retrieving the decryption information π_{ρ_a} that will become publicly available. This will reveal $\mathsf{degInfo}$ containing the signing tokens. The proxy signer can use any of these tokens to sign a message m. In order to do so, the proxy signer chooses a random e and computes a signature using any of the unused (k, w, z) in $\mathsf{degInfo}$. This produces the second layer Schnorr signature. As shown, this signature has the same structure as the signatures that the original signer would produce, and will be verified using the same Verify algorithm.

Revocation. We support decentralized and anonymous revocation that does not rely on any trusted/semi-trusted party, and does not reveal the proxy signer identity or that a delegation took place. We have two types of revocation: automatic, when the delegation period is over and it is enforced by the timed property discussed above, and on-demand, where the original signer can end the delegation early (before time ρ_b) using the trapdoor τ_b associated to the delegation. Both are done by decrypting ct_b and publishing all unused k values on the board, preventing the proxy signer from using them (recall in Verify any signature with a k value that is already on the board will be rejected). 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) will be used to $\mathsf{PreOpen}$ ct_b.[13]

Policy enforcement. The previous construction enforces a time policy but allows signing any message. To restrict the proxy signer to sign messages that satisfy certain policy policy_m, we can adopt two generic approaches from the literature [28, 34].

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 from the one used in the delegation (to prevent a proxy signer from using one of the signing tokens to sign any policy she wishes). So KeyGen will involve generating the other keypair used for policy signing and verification (note that the signature scheme for this one could be any secure signature scheme, no need to be a proxy scheme). Thus, each original signer in the system will be known using two public keys: the one for the policy and the one for the proxy signature. The original signer sends the signed policy as part of $\mathsf{degInfo}$. For Sign, the original signer can pick any policy_m, sign it, and have it as input to this algorithm. Both Sign and $\mathsf{DegSign}$ will output this public policy_m as part of θ. Verifying a signature will then be modified to include an additional check, which is verifying that policy_m is signed by the original signer, and that the signed message does not violate that policy.

Private policy. Here we can use a non-interactive zero knowledge (NIZK) proof system. A signature now will include a proof π attesting that the signed message m satisfies a private policy policy_m encoded as a function f (which represents the circuit C that encodes the required conditions). To make sure that f is chosen by the original signer, policy_m must be signed just like the warrant above. Thus, the NIZK proof π attests that some private policy_m (that only the proxy and original signers know) is signed by the original signer, and that m satisfies it. So, now a signature, whether produced by Sign or $\mathsf{DegSign}$, will include the proof π as part of θ, and 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. Also, the public parameters of the system will include any public parameters needed for the NIZK proof system.

Remark 1 (Threshold delegation). An interesting direction is to delegate signing rights to a committee rather than a single party [36], and require that at least t parties participate in signing a message. We can extend our techniques to construct a threshold proxy signature, in the sense that the delegation information will be distributed among n proxy signers and the proxy signing process will require t proxy signers to sign instead of one. We can use ideas from threshold Schnorr signature [32] to enable such threshold delegation. In Schnorr scheme, signatures

[13]Note that the original signer can alternatively store the klist associated with a delegation and simply reveal this list to execute on-demand revoke, i.e., no need to invoke $\mathsf{PreOpen}$. However, storing only the trapdoor, with a pointer to ct_b to locate it on the board, may lead to a smaller internal state at the original signer side.

are simply linear combinations of secret values, meaning that they are threshold-friendly by applying any of the homomorphically additive secret sharing schemes. For the signing tokens which correspond to a group element and its discrete logarithm, we can use the distributed key generation protocol (DKG) for discrete logarithm based systems by Gennaro et al. [33].

5 Design Considerations

Our signature delegation scheme relies on a public bulletin board to realize the time notion, the one-time property, and revocability. This may raise several issues in practice, which we discuss below along with solutions to handle them.

Invoking RoundBroadcast and decrypting ct_b. As mentioned in Section 2, we employ a decentralized TLE scheme in our construction. The decryption information is produced automatically for each round (either by relying on a period random beacon as in [30] or a committee that will be elected at round ρ as in [23]). This information will be publicly accessible, and we assume it 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 validators of the board (or simply the miners of the underlying blockchain) do. Thus a validator will keep a record of all ct_b for each round ρ, when π_ρ becomes available, it will decrypt ct_b and post all unused k values on the board. Furthermore, the definition of $\mathsf{RoundBroadcast}$ captured 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 [30], s will be shared among the producers of the threshold random beacon—each producer will obtain a share).

Mass publication of unused k values and anonymity. 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 on the board 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 for her own signatures as outlined in the Sign algorithm in Figure 5.

Denial of service attack against signers. As noted, a signature will be directly rejected if its k value is already published on the board. In the previous section, we state that the k values are either published by the honest verifier (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 denial of service attack against the original or proxy signers by intercepting a signature and publishing its k value on the board to invalidate the signature when verified.

We can address this attack by modifying our scheme 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, in this modified approach, ct_b will contain valid signatures over random hashes using the tokens instead of just a list of the k values of these tokens. This solution will lead to increased storage cost on the board; we leave optimizing that as a future work.

Off-chain processing risks. Recall that our scheme is similar to off-chain processing in blockchain-based systems. That is, a signature is handed directly to verifiers who rely on the current state of the board when verifying this signature, i.e., based on whether a valid signature using same k value over a different message is published. 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 this attack 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 a verifier finds out that more than one valid signature (over different messages than what she has) using the same k and Z values in θ appeared on the board, they will reject the signature.

Bulletin board synchronization issues. In the previous section we implicitly assume that the board is instantly synced. That is, any information that is sent to be published will appear directly and all verifiers 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. This is in a sense similar to the issue of off-chain processing above. Thus, the delayed signature acceptance technique can be used to handle this issue as well.

6 Security

Before providing the main result that states the security of our construction, we discuss the choice of the security model and some considerations raised by prior work. We also recall the security notions of the bulletin board and the NIZK scheme needed to prove security of revocability, timed delegation, and policy enforcement. This is in addition to the definition of the Schnorr knowledge of exponent assumption that we rely on while proving unforgeability of our proxy signature scheme.

As we aim to have our scheme usable by open-access distributed systems that do not assume a public key infrastructure (PKI), we account for eventual related key attacks (RKA) on Schnorr signatures and use the solution proposed in [47] to mitigate this vulnerability, i.e., by hashing the public key concatenated with the message to be signed as discussed in Section 2. Since RKA considers a broader class of attacks than ordinary attacks, security against RKA implies strong unforgeability for our resulting proxy signature.

Below we informally recall the security properties we require from the underlying public bulletin board and the NIZK system (the former is based on the security notion of blockchains, e.g., [50]).

Definition 3 (Security of the bulletin board). *A public bulletin board is secure if it satisfies two properties: persistence and liveness, which are (informally) defined as follows:*

Persistence. *For any two honest parties P_1 and P_2, and any two time rounds ρ_1 and ρ_2 such that $\rho_1 \leq \rho_2$, with overwhelming probability the confirmed state of the board maintained by party P_1 at time ρ_1 is a prefix of the confirmed state of the board maintained by party P_2 at time ρ_2.*

Liveness. *If information* info *is broadcast at time round ρ (to be published on the board), then with overwhelming probability* info *will be recorded on the board at time at most $\rho + v$, where v is the liveness parameter.*

Note that liveness includes growth and availability (i.e., new data is being added to the bulletin board and, this board is accessible by any party at any time), and quality (i.e., the board records only valid information produced by honest parties).

Definition 4 (Security of NIZK). *Let $\mathcal{R}$ be an efficiently computable binary relation which consists of pairs of the form (x, w), and let $L_{\mathcal{R}}$ be the language associated with $\mathcal{R}$. A secure non-interactive zero-knowledge proof system (NIZK) for $\mathcal{R}$ must satisfy three properties: completeness, computational soundness, and zero-knowledge, which are (informally) defined as follows:*

Completeness. *An honest verifier always accepts a proof generated by an honest prover for a valid statement x using a valid witness w.*

Soundness. *A* PPT *adversary can convince an honest verifier to accept a proof of an invalid statement x (i.e., an x that is not in $L_{\mathcal{R}}$) with at most a negligible probability.*

Zero-knowledge. *It is possible to simulate (in polynomial time) the honest prover for any instance $x \in \mathcal{L}_{\mathcal{R}}$ without knowing a witness w. This insures that a NIZK proof does not reveal any information about the witness beyond the validity of the statement.*

In our proof of ProxyEUF-CMA of our scheme, we rely on the EUF-CMA security of Schnorr signature (which holds under the discrete logarithm assumption in the random oracle model), and on the Schnorr knowledge of exponent assumption (denoted as schnorr-koe) introduced by Crites et al. [20]. The schnorr-koe assumption states that an adversary $\mathcal{A}$ that forges a Schnorr signature with respect to a public key of her choice must know the corresponding secret key. In other words, there exists an extractor Ext that when given the signature, the public key, and the random coins of $\mathcal{A}$, can extract the corresponding secret key which is basically computing the discrete logarithm of the public key. Crites et al. show that this assumption is true under the discrete logarithm assumption in the algebraic group model [27]. They define a security game for schnorr-koe, denoted as $\mathsf{Exp}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda)$, in which the adversary is challenged to produce a valid Schnorr signature under a public key of her choice, and the adversary wins the game if the extractor fails in extracting the secret key corresponding to this public key. We informally recall the definition of the schnorr-koe assumption below while full details can be found in [20].

Definition 5 (The schnorr-koe Assumption). *Let $\mathbb{G}$ by a cyclic group of order q and generator G in which the discrete logarithm assumption holds. The schnorr-koe assumption holds with respect to $\mathbb{G}$ if for any* PPT *adversary $\mathcal{A}$, there exists a* PPT *extractor Ext and a negligible function* negl *such that* $\Pr[\mathsf{Exp}^{\mathit{schnorr\text{-}koe}}_{\mathcal{A},\mathit{Ext}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Theorem 1. *Assuming the EUF-CMA security of the Schnorr signature scheme, the schnorr-koe assumption, a secure bulletin board, a CCA secure TLE scheme, a EUF-CMA secure signature scheme Π, and a secure NIZK proof system Σ, the construction described in Figures 5 and 6 is an anonymous, timed and revocable proxy signature scheme (cf. Definition 2).*

Correctness is immediate by the correctness of the Schnorr signature scheme, correctness and availability of the bulletin board, correctness of TLE (that allows the proxy signer to correctly decrypt degInfo), and correctness of the building blocks used for policy enforcement.

For security, we show that our construction satisfies the security properties defined in Section 3 as follows.

Lemma 1 (Unforgeability). *Assuming EUF-CMA security of Schnorr signature scheme and the schnorr-koe assumption, the construction described in Figures 5 and 6 satisfies ProxyEUF-CMA security as defined in Figure 1 in the random oracle model and the algebraic group model.*

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the ProxyEUF-CMA security (based on the game defined in Figure 1) of our construction. For the sake of contradiction, consider that $\mathcal{A}$ can break ProxyEUF-CMA security with non-negligible probability. This means that $\mathcal{A}$ can produce a valid proxy signature $(\sigma^* = (w^*, c^*, s^*), \theta^* = (k^*, Z^*))$ over m^* such that $O\mathsf{Sign}$ was not queried over m^*, and $O\mathsf{Delegate}$ did not produce the token (w^*, k^*, z^*) where z^* is such that $Z^* = G^{z^*}$.

In a nutshell, the success of this forgery means that $\mathcal{A}$ is able to output a valid token, i.e., layer 1 Schnorr signature, and use this token to produce a valid proxy signature, i.e., layer 2 Schnorr signature. Thus, for the latter there exists an Ext that can extract z^* from this signature and the public key Z^* such that $Z^* = G^{z^*}$, if not, then this means that $\mathcal{A}$ is able to break the $\mathsf{schnorr\text{-}koe}$ assumption (which is a contradiction). Then, if extraction succeeds, we have that (z^*, w^*) is a forgery for the message k^* with respect to layer 1, breaking the EUF-CMA security of Schnorr signature with respect to the original signer's key vk (which is also a contradiction).

More formally, we construct two PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$. $\mathcal{B}_1$ attempts to break the EUF-CMA security of Schnorr signature with respect to vk, while $\mathcal{B}_2$ attempts to break the $\mathsf{schnorr\text{-}koe}$ assumption with respect to Z^* produced by $\mathcal{A}$. As a result, let the advantage of $\mathcal{B}_1$ be $\mathsf{Adv}_{\mathcal{B}_1} = \Pr[\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Schnorr},\mathcal{B}_1}(\lambda) = 1]$, and the advantage of $\mathcal{B}_2$ be $\mathsf{Adv}_{\mathcal{B}_2} = \Pr[\mathsf{Exp}^{\mathsf{schnorr\text{-}koe}}_{\mathsf{Schnorr},\mathcal{B}_2}(\lambda) = 1]$, we have:

$$\Pr[\mathsf{Exp}^{\mathsf{ProxyEUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{Adv}_{\mathcal{B}_1} + \mathsf{Adv}_{\mathcal{B}_2} + \mathsf{negl}(\lambda)$$

Adversary $\mathcal{B}_1$ has access to the signing oracle $O\mathsf{Sign}^{\mathsf{Schnorr}}_{\mathsf{vk}}(\cdot)$ and should produce a forgery on a new message $m^* \notin \mathsf{L}^{\mathsf{Schnorr}}_{\mathsf{sign}}$, i.e., not queried before to the signing oracle. It does that by invoking $\mathcal{A}$ to produce a forged proxy signature (based on the game defined in Figure 1), then it invokes the extractor Ext in the $\mathsf{schnorr\text{-}koe}$ game that $\mathcal{B}_2$ is attempting to break to extract z^* from the Z^* part in the forged proxy signature. This z^* serves as a forged Schnorr signature over message k^* against vk. If the extractor fails, i.e., it cannot compute the discrete logarithm z^* of Z^*, this means that $\mathcal{B}_2$ can use $\mathcal{A}$'s output to break the $\mathsf{schnorr\text{-}koe}$ assumption.

$\mathcal{B}_1$ simulates $\mathcal{A}$ oracle queries in Figure 1 as follows. When $\mathcal{A}$ requests $O\mathsf{Sign}(m)$, $\mathcal{B}_1$ answers as follows:

1. Update list $\mathsf{L}_{\mathsf{sign}} \leftarrow \mathsf{L}_{\mathsf{sign}} \cup \{m\}$
2. Sample a random $k \in \mathbb{Z}_q$
3. Query $(z, w) \leftarrow O\mathsf{Sign}^{\mathsf{Schnorr}}_{\mathsf{vk}}(k)$
4. Add k to $\mathsf{L}^{\mathsf{Schnorr}}_{\mathsf{sign}}$
5. Sample a random $e \in \mathbb{Z}_q$
6. Compute $E = G^e, Z = G^z$
7. Compute $c = \mathsf{H}(m, Z, E)$
8. Compute $s = (e + cz) \mod q$
9. Return $(\sigma = (w, c, s), \theta = (k, Z))$

When $\mathcal{A}$ requests $O\mathsf{Delegate}(\mathsf{vk})$, $\mathcal{B}_1$ answers as follows:[14]

1. Sample a random $k \in \mathbb{Z}_q$
2. Query $(z, w) \leftarrow O\mathsf{Sign}^{\mathsf{Schnorr}}_{\mathsf{vk}}(k)$
3. Update list $\mathsf{L}_{\mathsf{deleg}} \leftarrow \mathsf{L}_{\mathsf{deleg}} \cup \{(\mathsf{degInfo} = (z, w, k)\}$
4. Return $\mathsf{degInfo} = (z, w, k)$

[14]Note that, to simplify the presentation and without loss of generality, we let $O\mathsf{Delegate}$ generate one signing token for every query instead of generating several tokens based on $\mathsf{degspec}$ that $\mathcal{A}$ can choose. We also do not include a time policy, which is also a part of $\mathsf{degspec}$, since we prove security of our timed notion when proving the policy enforcement property in Lemma 4.

The adversary $\mathcal{A}$ produces a forgery $(\sigma^* = (w^*, c^*, s^*), \theta = (k^*, Z^*))$ on a new message $m^* \notin \mathsf{L}_{\mathsf{sign}}$ and new $\theta = (k^*, Z^*) \notin \mathsf{L}_{\mathsf{deleg}}$ that was not queried to $O\mathsf{Delegate}$. By the security of the $\mathsf{schnorr\text{-}koe}$ assumption, $\mathcal{B}_1$ can obtain the discrete log z^* of Z^*. This holds since the second layer of our proxy signature, namely (c^*, s^*) is a Schnorr signature over m^* with respect to the key Z^* that the adversary $\mathcal{A}$ chooses. Otherwise, meaning that if extraction fails, then $\mathcal{B}_2$ can use (c^*, s^*, Z^*) to break the $\mathsf{schnorr\text{-}koe}$ assumption, which is a contradiction as the advantage of $\mathcal{B}_2$ in doing that is negligible.

Now, since the extractor will succeed, $\mathcal{B}_1$ outputs the message k^* and the signature $\sigma^* = (z^*, w^*)$ as a valid forgery to Schnorr signature for the verification key vk. Given that Schnorr signature is EUF-CMA secure, the advantage of $\mathcal{B}_1$ is also negligible. Thus, $\mathcal{A}$'s advantage in producing a valid forgery against our proxy signature scheme is negligible, which completes the proof. □

Lemma 2 (Anonymity). *The construction described in Figures 5 and 6 satisfies the anonymity property defined in Figure 2.*

Proof. In the $\mathsf{DegAnon}$ security game, defined in Figure 2, $\mathcal{A}$ may rely on the following to distinguish the challenge signature:

1. **Case 1: signature structure.** $\mathcal{A}$ tries to use $(\bar{\sigma}, \bar{\theta})$ to infer any information on whether the original or proxy signer has produced this signature.
2. **Case 2: bulletin board state.** $\mathcal{A}$ monitors the bulletin board to tell if a new delegation has been created, and thus case $b = 0$ was executed by the challenger. This allows $\mathcal{A}$ to guess correctly that the $(\bar{\sigma}, \bar{\theta})$ was produced by a proxy signer.

For case 1, note that in our construction the two signatures (σ_0, θ_0) and (σ_1, θ_1) are identically distributed, and hence, indistinguishable for an outsider. Both have an identical structure and are verified in the same way against the original signer's verification key vk. Thus, $(\bar{\sigma}, \bar{\theta})$ will not provide $\mathcal{A}$ with any information on whether this signature was produced by Sign or $\mathsf{DegSign}$.

For case 2, note that $\mathsf{degInfo}$ in our construction, in particular the portion of the time policy published on the board—(ρ_b, ct_b), does not reveal any information about the signing tokens, which delegation they are tied to (if any), or the identity of the proxy signer. Moreover, the *indistinguishable behaviour* of the original signer ensures hiding that a delegation took place. That is, the original signer simulates a "self-delegation" and behaves in a similar way to the case of delegation even when generating signatures using her signing key—by posting dummy (ρ_b, ct_b) simulating a timed delegation or fresh klist simulating a delegation revocation. The original signer will also include policy policy_m with her signatures. This will make Sign and $\mathsf{DegSign}$ follow the same behavior. Thus, seeing time policies on the board, or published k values, or signatures with policies over the signed messages, will not give the adversary $\mathcal{A}$ any additional advantage in distinguishing the signatures.

This is in addition to the fact that delegation in our construction is generated in a non-interactive way, i.e., the original signer does that alone. Also, the generated delegation information is sent securely to the proxy signer. So, only these two parties are aware that a delegation took place while any information posted publicly on the board does not reveal that.

As a result, the two strategies above will not give the attacker $\mathcal{A}$ any additional advantage, over just random guessing, in winning the $\mathsf{DegAnon}$. This completes the proof. □

Lemma 3 (Revocability). *Assuming a secure bulletin board and a CCA secure TLE scheme, then the construction described in Figures 5 and 6 satisfies the revocability property defined in Figure 3.*

Proof. Security of both automatic and on-demand revocability relies on the security of the bulletin board and the TLE scheme used in our construction. A verifier will check the board to validate the $\mathsf{revState}$ before accepting a valid signature, and will reject any signature that is already revoked. An adversary $\mathcal{A}$ who tries to break the revocability of our construction can do that by: (1) manipulating the bulletin board (i.e., prevent publishing $\mathsf{revState}$ on the board, control the verifier's view of the board—so the verifier sees the old state that does not contain the updated $\mathsf{revState}$, or even rewrite the board to omit already recorded $\mathsf{revState}$), or (2) maul (ρ_b, ct_b) to produce different $\mathsf{revState}$ (or k values) from those produced by the original signer.

By the assumption that the bulletin board is secure (cf. Definition 3), the first strategy succeeds with negligible probability. Also, by the assumption that the TLE scheme we use in our construction is CCA secure, the second strategy succeeds with negligible probability. Thus, $\mathcal{A}$'s advantage in breaking the revocability property of our scheme is negligible, which completes the proof. □

Lemma 4 (Policy enforcement). *Assuming a secure bulletin board, a CCA secure TLE scheme, an EUF-CMA secure signature scheme Π, and a secure NIZK system Σ, then the construction described in Figures 5 and 6, with the policy enforcement techniques described in Section 4, satisfies the policy enforcement property defined in Figure 4.*

Proof. Recall that in our scheme a policy is composed of two parts: $\mathsf{policy} = (\mathsf{policy}_t, \mathsf{policy}_m)$. We show that our construction satisfies policy enforcement for both of them.

For the time policy policy_t, security relies on the security of the bulletin board and the TLE scheme. By the correctness of the TLE scheme, the valid decryption information will be produced at times ρ_a and ρ_b. Thus, the proxy signer can reveal the $\mathsf{degInfo}$ only when time ρ_a comes, and the board validators can decrypt ct_b only when time ρ_b comes. By Lemma 3, an adversary $\mathcal{A}$ cannot prevent automatic revocation, which ends the delegation time period at time ρ_b. In other words, an adversary that can violate policy_t can be used to build another adversary $\mathcal{B}$ that can break the security of the TLE scheme and the revocability of our construction, which is a contradiction.

For the message policy policy_m, we discuss both public and private policy options in our construction. For public policy_m, its security relies on the EUF-CMA security of the signature scheme Π that the original signer uses to sign the public warrant. A verifier will verify this signature first before checking that $m \in \mathsf{policy}_m$. An adversary $\mathcal{A}$ cannot forge a valid signature over a policy of her choice with non-negligible probability. For private policy_m, using a similar argument, an adversary $\mathcal{A}$ cannot forge a signature over a policy_m of her choice to produce a valid NIZK. And by the security of the NIZK system (cf. Definition 4), $\mathcal{A}$ cannot produce a valid proof that convinces the verifier of accepting a signature over a message $m \notin \mathsf{policy}_m$ (where here policy_m is a valid policy produced by the original signer). Furthermore, the integrity of a valid NIZK proof π is preserved since in our construction the produced signature in case of private message policy is over $m \parallel \pi$, which completes the proof. $\square$

Proof of Theorem 1. Follows by Lemmas 1, 2, 3, and 4. $\square$

References

1. Abadi, A., Ciampi, M., Kiayias, A., Zikas, V.: Timed signatures and zero-knowledge proofs—timestamping in the blockchain era—. In: International Conference on Applied Cryptography and Network Security. pp. 335–354. Springer (2020)
2. Abe, M., David, B., Kohlweiss, M., Nishimaki, R., Ohkubo, M.: Tagged one-time signatures: Tight security and optimal tag size. In: The 16th International Conference on Practice and Theory in Public-Key Cryptography (PKC). pp. 312–331. Springer (2013)
3. Amos, R., Georgiou, M., Kiayias, A., Zhandry, M.: One-shot signatures and applications to hybrid quantum/classical authentication. In: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing. pp. 255–268 (2020)
4. Arun, A., Bonneau, J., Clark, J.: Short-lived zero-knowledge proofs and signatures. Cryptology ePrint Archive (2022)
5. Ateniese, G., Hohenberger, S.: Proxy re-signatures: new definitions, algorithms, and applications. In: Proceedings of the 12th ACM conference on Computer and communications security. pp. 310–319 (2005)
6. Backes, M., Meiser, S., Schröder, D.: Delegatable functional signatures. In: Public-Key Cryptography–PKC 2016, pp. 357–386. Springer (2016)
7. Bellare, M., Goldwasser, S.: Encapsulated key escrow (1996)
8. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In: The International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 491–506. Springer (2003)
9. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: The 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 409–426. Springer (2006)
10. Ben-David, S., Sattath, O.: Quantum tokens for digital signatures. arXiv preprint arXiv:1609.09047 (2016)
11. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: International conference on the theory and applications of cryptographic techniques. pp. 127–144. Springer (1998)
12. Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. Journal of Cryptology **25** (06 2003). `https://doi.org/10.1007/s00145-010-9082-x`

13. Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. Journal of Cryptology **25**(1), 57–115 (2012)
14. Boneh, D., Naor, M.: Timed commitments. In: The 20th Annual International Cryptology Conference (CRYPTO). pp. 236–254. Springer (2000)
15. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: International workshop on public key cryptography. pp. 501–519. Springer (2014)
16. Campanelli, M., David, B., Khoshakhlagh, H., Konring, A., Nielsen, J.B.: Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. Cryptology ePrint Archive (2021)
17. Chalkias, K., Hristu-Varsakelis, D., Stephanides, G.: Improved anonymous timed-release encryption. In: The 12th European Symposium On Research In Computer Security (ESORICS). pp. 311–326. Springer (2007)
18. Choi, G., Vaudenay, S.: Timed-release encryption with master time bound key (full version). Cryptology ePrint Archive (2019)
19. Coladangelo, A., Liu, J., Liu, Q., Zhandry, M.: Hidden cosets and applications to unclonable cryptography. In: The 41st Annual International Cryptology Conference (CRYPTO). pp. 556–584. Springer (2021)
20. Crites, E., Komlo, C., Maller, M.: How to prove schnorr assuming schnorr: security of multi-and threshold signatures. Cryptology ePrint Archive (2021)
21. Das, M.L., Saxena, A., Gulati, V.P.: An efficient proxy signature scheme with revocation. Informatica **15**(4), 455–464 (2004)
22. Derler, D., Hanser, C., Slamanig, D.: Privacy-enhancing proxy signatures from non-interactive anonymous credentials. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 49–65. Springer (2014)
23. Döttling, N., Hanzlik, L., Magri, B., Wohnig, S.: Mcfly: Verifiable encryption to the future made practical. Cryptology ePrint Archive (2022)
24. Even, S., Goldreich, O., Micali, S.: On-line/off-line digital signatures. In: Advances in Cryptology—CRYPTO. pp. 263–275. Springer (1990)
25. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology—CRYPTO. vol. 86, pp. 186–194. Springer (1986)
26. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: Proceedings of the 5th ACM Conference on Computer and Communications Security. pp. 83–92 (1998)
27. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: The 38th Annual International Cryptology Conference (CRYPTO). pp. 33–62. Springer (2018)
28. Fuchsbauer, G., Pointcheval, D.: Anonymous proxy signatures. In: International Conference on Security and Cryptography for Networks. pp. 201–217. Springer (2008)
29. Fuchsbauer, G., Pointcheval, D.: Anonymous consecutive delegation of signing rights: Unifying group and proxy signatures. Formal to Practical Security: Papers Issued from the 2005-2008 French-Japanese Collaboration pp. 95–115 (2009)
30. Gailly, N., Melissaris, K., Romailler, Y.: tlock: Practical timelock encryption from threshold bls. Cryptology ePrint Archive, Paper 2023/189 (2023), `https://eprint.iacr.org/2023/189`
31. Garay, J.A., Jakobsson, M.: Timed release of standard digital signatures. In: The 6th International Conference on Financial Cryptography and Data Security. pp. 168–182. Springer (2003)

32. Garillot, F., Kondi, Y., Mohassel, P., Nikolaenko, V.: Threshold Schnorr with stateless deterministic signing from standard assumptions. pp. 127–156 (2021). `https://doi.org/10.1007/978-3-030-84242-0_6`
33. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. pp. 295–310 (1999). `https://doi.org/10.1007/3-540-48910-X_21`
34. Hanser, C., Slamanig, D.: Blank digital signatures. In: The 8th ACM SIGSAC symposium on Information, computer and communications security. pp. 95–106 (2013)
35. Hanser, C., Slamanig, D.: Warrant-hiding delegation-by-certificate proxy signature schemes. In: International Conference on Cryptology in India. pp. 60–77. Springer (2013)
36. Herranz, J., Sáez, G.: Verifiable secret sharing for general access structures, with application to fully distributed proxy signatures. In: International Conference on Financial Cryptography. pp. 286–302. Springer (2003)
37. Kim, S., Park, S., Won, D.: Proxy signatures, revisited. In: International Conference on Information, Communications and Signal Processing (1997)
38. Lamport, L.: Constructing digital signatures from a one way function (1979)
39. Lee, B., Kim, H., Kim, K.: Strong proxy signature and its applications. In: Proceedings of SCIS. vol. 2001, pp. 603–608 (2001)
40. Lee, J.Y., Cheon, J.H., Kim, S.: An analysis of proxy signatures: Is a secure channel necessary? In: The Cryptographers' Track at the RSA Conference, CT-RSA. pp. 68–79. Springer (2003)
41. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Designs, Codes and Cryptography **86**, 2549–2586 (2018)
42. Liu, Z., Hu, Y., Zhang, X., Ma, H.: Provably secure multi-proxy signature scheme with revocation in the standard model. Computer Communications **34**(3), 494–501 (2011)
43. Lu, E.J.L., Hwang, M.S., Huang, C.J.: A new proxy signature scheme with revocation. Applied mathematics and Computation **161**(3), 799–806 (2005)
44. Malkin, T., Obana, S., Yung, M.: The hierarchy of key evolving signatures and a characterization of proxy signatures. In: The International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 306–322. Springer (2004)
45. Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: Proceedings of the 3rd ACM Conference on Computer and Communications Security. pp. 48–57 (1996)
46. Mambo, M., Zheng, Y., Okamoto, T., Tada, M., Okamoto, E.: Extended proxy signatures for smart cards. In: Information Security Workshop, ISW. pp. 247–258. Springer (1999)
47. Morita, H., Schuldt, J.C., Matsuda, T., Hanaoka, G., Iwata, T.: On the security of the schnorr signature scheme and dsa against related-key attacks. In: The 18th International Conference (ICISC). pp. 20–35. Springer (2016)
48. Neuman, B.: Proxy-based authorization and accounting for distributed systems. In: [1993] Proceedings. The 13th International Conference on Distributed Computing Systems. pp. 283–291 (1993)
49. Okamoto, T., Tada, M., Okamoto, E.: Extended proxy signatures for smart card. In: Lecture Notes in Computer Science book series (LNCS,volume 1729). pp. 247–258. Springer Berlin Heidelberg (1999)

50. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 643–673. Springer (2017)
51. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**(3), 361–396 (2000)
52. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
53. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Advances in Cryptology—CRYPTO. pp. 239–252. Springer (1990)
54. Schuldt, J.C., Matsuura, K., Paterson, K.G.: Proxy signatures secure against proxy key exposure. In: The 11th International Workshop on Practice and Theory in Public-Key Cryptography (PKC). pp. 141–161. Springer (2008)
55. Sun, H.M.: Design of time-stamped proxy signatures with traceable receivers. IEE Proceedings-Computers and Digital Techniques **147**(6), 462–466 (2000)
56. Sun, H.M., Lee, N.Y., Hwang, T.: Threshold proxy signatures. IEE Proceedings-Computers and Digital Techniques **146**(5), 259–263 (1999)
57. Thyagarajan, S.A., Malavolta, G., Schmid, F., Schröder, D.: Verifiable timed linkable ring signatures for scalable payments for monero. In: The 27th European Symposium on Research in Computer Security (ESORICS). pp. 467–486. Springer (2022)
58. Varadharajan, V., Allen, P., Black, S.: An analysis of the proxy problem in distributed systems. In: Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy. pp. 255–275 (1991)
59. Wang, H., Pieprzyk, J.: Efficient one-time proxy signatures. In: The 9th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 507–522. Springer (2003)
60. Wuille, P., Nick, J., Ruffing, T.: Schnorr signatures for secp256k1. Bitcoin Improvement Proposal 340 (2020), `https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki`
61. Xu, S., Yang, G., Mu, Y., Ma, S.: Proxy signature with revocation. In: Australasian Conference on Information Security and Privacy. pp. 21–36. Springer (2016)
62. Zhang, F., Kim, K.: Efficient id-based blind signature and proxy signature from bilinear pairings. In: The 8th Australasian Conference (ACISP). pp. 312–323. Springer (2003)