Policy-Based Access Tokens: Privacy-Preserving Verification for Digital Identity

Kiran Pun, Daniel Gardham✉ and Nick Frymann

Surrey Centre for Cyber Security, University of Surrey, United Kingdom
✉ daniel.gardham@surrey.ac.uk

Abstract. Passports, driving licences, and other government-issued identity documents are frequently used to prove attributes about an individual, such as their date of birth or home address. Traditional paper-based approaches are being transitioned to *digital identities*, which are becoming increasingly important for online interactions and transactions, allowing individuals to prove their identity without needing to present physical documents. However, existing solutions suffer from cumbersome primitives, for example, the European Commission is actively experimenting with Zero-Knowledge proof based solutions for the EU's Digital Identity Wallet, or lack of functionality such as the UK's right-to-work share codes.

In this paper, we present a new cryptographic primitive, Policy-Based Access Tokens, that allows for lightweight verification of user attributes through a service (such as a government office). We propose two variants of the scheme: PAT-I offers token unforgeability such that malicious parties cannot verify personal data without a valid token. This is then extended in PAT-II to allow for distributed delegation to a set of proxies, offering fine-grained revocation. We consider stronger security properties that prevent proxies colluding, whilst providing anonymity against the service provider. We give generic constructions of our schemes, prove their security in the standard model, and provide instantiations based on bilinear pairings. Finally, we provide a proof-of-concept implementation which demonstrates that our protocols are efficient, with token verification taking $\approx$ 100ms.

Keywords: Digital Identity · Anonymous Tokens · Delegation.

1 Introduction

Government-issued identity documents, such as passports or driving licences, are often used to prove attributes about a person, such as name, date of birth, or home address. Traditionally, this would require presenting physical documents, but in doing so, would reveal the entire set of information contained within, often far exceeding that which is necessary. For example, a customer proving their age should not need to share their residential address when purchasing age-restricted products, as is typically the case with e.g., national identity cards.

Digital services are replacing traditional paper-based approaches. The EU's Digital Identity (EUDI) Wallet [15] allows individuals to share their identity data online and offline for both public and private services based on the EUDI Architecture and Reference Framework (ARF). Users can collect attributes from various services and present tokens to verifiers when proving attributes—this verification step can be offline to the issuer. Whilst user control is a core principle of the design, misuse or leakage of credentials by dishonest verifiers can leak information to an unwanted third party and it does not support native time-limited verification.

Furthermore, currently used formats in the ARF such as SD-JWT (Selective Disclosure JSON Web Token) and ISO mDL (mobile Driver's License), which rely on salted hashes, do not fully eliminate linkability concerns. Whilst using Zero-Knowledge proofs (ZKPs) to address this is under active development and discussion in the ARF, they are nonetheless known to be computationally complex and often inefficient. This complexity, coupled with the lack of standardised support in currently available secure hardware (i.e., Wallet Secure Cryptographic Devices, WSCDs), affects adoption of ZKPs in the EUDI Wallet ecosystem [23].

A lighter-weight approach is used in the United Kingdom in its Right-to-Work and Right-to-Rent share code schemes for non-British and non-Irish citizens [14, 21]. The Home Office creates a unique, nine-character alphanumeric code to enable these individuals to digitally prove their immigration status, as well as their right to work (for employers) or right to rent (for landlords). This system streamlines the verification process, replacing the need for physical documents and allows for the verification to be time-bounded, typically valid for 90 days.

This approach requires the service provider to ensure validity of the attribute at time of verification, rather than issuance, through an online check. However, the richness of this scheme is limited as it can only be used to check the types of work an individual can undertake and for how long. Similar approaches are used for the UK's Disclosure and Barring Service (DBS) [20] to prove safety compliance. We further note the recent announcement by the UK to adopt Digital ID [35], but technical details are currently unknown.

Extending the functionality of share codes to prove additional personal information, e.g., those typically found on identity documents, would better enable digital identity. In such a setting, governments are expected to act as a high-availability, centralised, and trusted service provider. This trust and availability could be leveraged to omit the complexity challenges encountered by ZKP or anonymous credential-based approaches by providing online verification to real-time queries. However, there are two main challenges with this naïve approach.

Firstly, due to the lack of fine-grained control, a verifier could view any data field of its choosing, thus breaching the user's privacy. This is the aforementioned drawback of a paper-based approach, where selective disclosure of data is not possible. Therefore, any solution would need to provide a user with fine-grained control over which information can be accessed, encoded in a policy. This aligns with the strong privacy controls offered by the EU digital wallet, and could also enable additional constraints such as time or role-based policies (enforced by the

service provider). Secondly, due to the semi-static nature of the share codes, and in particular, that it is not unique to the verifier, the code is identical for all queries within a refresh window. Thus, a verifying party (which we call a *proxy*) that intentionally or otherwise leaks a share code, would enable malicious parties to query and ultimately learn personal information.

Whilst shorter refresh windows limit exposure, they also increase overhead for users and the issuing service. Therefore, it is desirable that codes are bound to unique verifiers, and can be revoked at any time (by the user). However, this introduces additional privacy concerns as token redemption may leak additional information about the services with which users interact. For a centralised service such as a government ID scheme, this could leak behavioural patterns beyond what is intended by the user through analysis of meta-data. To prevent such attacks, any such scheme should also ensure anonymity of the proxies.

Contributions. In this paper we propose a new primitive, Policy-Based Access Tokens, that solves the challenges of extending share codes for use in wider digital identity environments. It enables users to delegate tokens that allow proxy verifiers to validate only selected information with a service.

In particular, we present two variations. PAT-I allows users to delegate tokens to proxies that are able to verify data specified in the policy. We define the *token unforgeability* security property which captures the notion that any token passing verification must have originated from an honest delegation by the user, thus preventing malicious proxies from querying the service provider without the user's permission.

We then extend this scheme in PAT-II to enable strong user control over the tokens by distributing trust over a set of designated proxies. In particular, this scheme prevents token sharing. To allow for fine-grained proxy management, we provide the user with a revocation mechanism, utilising a verifier-local approach for improved efficiency. Furthermore, this scheme offers strong privacy guarantees that we capture in the *proxy anonymity* property which prevents service providers from linking delegated tokens.

We present generic constructions for PAT-I and PAT-II based on standard cryptographic primitives and prove that they meet token unforgeability—and proxy anonymity for the latter. We provide a proof-of-concept Python implementation and show that, even without optimisations, our protocols are very efficient, with credential delegation taking approximately 500ms and verification around 100ms for both schemes.

Finally, we observe that our Policy-Based Access Tokens scheme may find uses in other applications, which may be of independent interest. For example, when combined with secure key derivation functions, our protocol can be adopted for use in offline file/disk encryption to allow temporary access without needing to re-encrypt under new keys (a potentially computationally intensive task).

Related work. The United Kingdom has introduced the Digital Verification Services (DVS) register, which operates under the Digital Identity and Attributes Trust Framework (DIATF) [22]. Certified providers are authorised to perform

identity verification in regulated contexts such as tenancy applications and criminal record checks [22]. The DIATF is technology-neutral and emphasises governance, assurance levels, and compliance, whilst leaving cryptographic choices to providers. This creates space for lighter-weight cryptographic methods, such as the one proposed in this work, to improve privacy within DIATF-compliant services. In particular, it provides a privacy-preserving way to check validity and authorisation without needing to see or store raw identity data, which keeps them aligned with DIATF's focus on user control.

Anonymous Credentials (ACs), first proposed by Chaum [9], allow users to present an attribute-based credential, such that a verifier is convinced that it has been issued to the user by a valid party—ACs could be adopted to present selective attributes to verifiers. One benefit of such an approach is that the verification step does not require the issuer to be online. However, ACs can suffer from inefficient complexity, typically due to their reliance on costly Zero-Knowledge proofs used to achieve unlinkability and anonymity (e.g., [3, 6]), although, we note other approaches rely on self-blinding techniques (as instigated by Verheul [42]).

Furthermore, credential updates require reissuing of credentials, and revocation (from the view of the issuer) is challenging. As summarised by Kakvi *et al.* [25], common methods include certificate-based and verifier-local approaches, both of which require the signer or verifier to maintain up-to-date lists, which can be cumbersome in practice. Nonetheless, a state-of-the-art scheme Server-Aided Anonymous Credentials (SAAC) by Chairattana-Apirom *et al.* [8] enables lightweight instantiations of publicly verifiable and multi-use ACs suitable compliance with existing national standards; candidate solutions [13] for the EUDI Wallet can be viewed similarly to SAAC. However, they require additional auxiliary information generated by the issuer, before verification can take place, thus adding an extra round of communication.

Chameleon hash functions [28] allow users, with knowledge of a trapdoor, to compute collisions and have been used in many credential schemes (e.g. [10, 27, 36]). Introducing policies to Chameleon hash functions have previously been explored by Tian *et al.* [40], who apply their primitive to achieve anonymity and accountability in secure blockchain rewriting, which has been extended to support revocation [43]. Note that here the *policy* refers to that of the trapdoor holder, i.e., a trapdoor owner who satisfies the access policy can amend the underlying message without affecting the hash value, rather than on verification.

Santizable signatures, introduced by Ateniese *et al.* [1], allow for selective disclosure of components of the message, which could be conceivably used to only share user-controlled information with a verifying party. However, the signature size is linear in the number of messages signed and thus the communication between the sanitizer and verifier is potentially orders of magnitude larger than what is required. Furthermore, there is no mechanism to enforce a verifying policy on the signature, such as a date. Revocation has been explored in the context of certificated sanitizable signatures [31], where revocation is achieved through periodic key updates. We note that policy-based sanitizable signatures have been

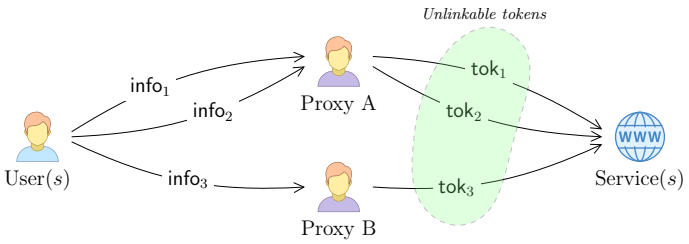


Figure 1: Our *proxy anonymity* property prevents service providers from correlating users through proxies, which could leak metadata, such as user behaviours.

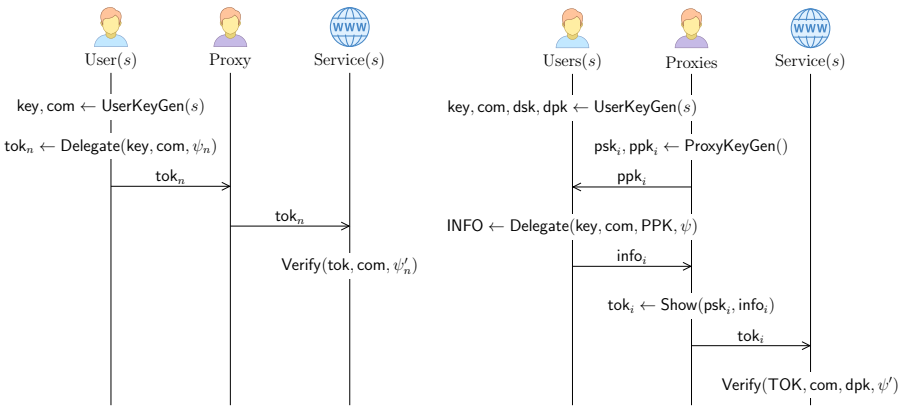
proposed by Samelin and Slamanig [37], where the policy is on the sanitization rights, not verification.

Designated-verifier signatures (DVS)—not to be confused with the Digital Verification Service—first formalised by Jakobsson, Sako, and Impagliazzo [24], ensures that only a specific designated verifier can validate a signature, whilst preventing them from convincing others of its authenticity. This non-transferability is achieved by enabling the verifier to simulate valid-looking signatures themselves. Steinfeld *et al.* [39] generalised this idea with universal designated-verifier signatures (UDVS), allowing any holder of a signature to designate it to a verifier using the verifier's public key, without the signer's involvement. This concept has also been applied to anonymous credentials, such as in the RETRACT framework [12], where credential proofs are bound to a designated verifier to prevent misuse or data leakage. This would be unsuitable for our setting as the verifier must be fixed during the setup phase, as the token is bound to a specific verifier from the outset.

2 Our Policy-Based Access Tokens Scheme

In this section, we formally describe our Policy-Based Access Tokens scheme. We give the syntax and security definitions for two variations of the scheme: PAT-I and the extended PAT-II, which supports issuance to sets of proxies and their granular revocation.

An overview of the protocol for our Policy-Based Access Tokens scheme is provided in Figure 2, which uses syntax from Section 2.1. In both PAT-I and PAT-II, the user holds a shared secret with the service provider, which could be an existing account credential, share code, or identity number—for example, a driving licence number (for the DVLA in the UK), to prove the user's identity (name) and the fact they hold a valid licence, or an insurance policy number, to prove the user has correct coverage. This shared secret is used to generate multiple tokens for proxy users, which allows verification of attributes of the user, without giving access to the secret itself. A policy ψ could specify a restricted timeframe for access (specific dates, or a single session), what data may be accessed (for example, name, home address, licence validity, account balance,



(a) Protocol flow for PAT-I. (b) Protocol flow for PAT-II.

Figure 2: The Policy-Based Access Tokens protocol flows. In PAT-I, the proxy simply passes tok to the service provider to gain access. To achieve revocation and the required anonymity property, PAT-II uses a proxy key ppk, which may be reused to generate tokens many times.

etc.), or permissions the proxy has upon presenting the token (e.g., read/write, read-only).

A policy ψ is encoded in tok and is communicated to the server as part of the verification request when the proxy presents tok. The applicable ψ is therefore uniquely determined by the contents of presented tok. The server derives a context-specific interpretation ψ' by applying parameters to the current verification context. The resulting interpretation ψ' is used to determine the scope, validity period, and permissions associated with the token. For example, it may create ψ' based on ψ by inserting the current time, or an age or other attribute of the user. These context-specific parameters ultimately determine the validity of the request. In line with our design goals, policies are intentionally restricted to simple, policy-specific conditions (e.g., conjunctive statements); more expressive or complex policy languages are not considered. The exact format of the policy is application dependent, as the types of permissions encoded may vary between use cases; although some standardisation could be introduced. In PAT-II, where a user can issue tokens to sets of proxies that must collaborate during the verification phase, the policy also describes the access structure for the proxies, e.g., conjunctive, threshold, monotone. We give a simple example in Example 1.

In PAT-I, summarised in Figure 2a, the user can generate many tokens for proxy users, who need only present them directly to the service provider. The user calls UserKeyGen with the secret s to generate a key key and commitment com. This is then used as input to Delegate which may generate many tokens tok; these are then distributed to proxies accordingly. The tokens tok_i are presented to the service provider who, using Verify, confirm its validity, which includes checks that the policy is satisfied.

To permit fine-grained user control and distributed delegation, in PAT-II (presented in Figure 2b), we allow the user to designate a particular set of proxies, which use proxy keys to generate unlinkable and revocable tokens. These can be combined in the verification step to ensure the set of proxies is valid with respect to the policy ψ, which encodes both a description of the access policy (e.g., the threshold of proxies) and other constraints e.g., date or read/write permissions.

In this scheme, the proxies generate and provide the user with a long-term proxy public key ppk. The user runs Delegate using a ppk to generate info, which includes a derived public key (which is authenticated by the proxy). The user may run Delegate many times to allow the proxy to access multiple service providers or to generate multiple access structures for distinct sets of proxies. The user provides info to the proxy, who calls Show with their long-term secret key psk to generate a token share tok to present to the service provider. Once a candidate set of tokens has been collected, using the Verify algorithm, the server confirms their validity against the policy. Proxies can be revoked at any time, with the user registering the derived public key ppk on a blocklist at the service provider.

The tokens provided by proxies offer *proxy anonymity* such that the service provider cannot determine the identity of the proxy, or even whether two tokens are for the same proxy. This prevents metadata leakage which could otherwise undermine users' privacy, such as disclosing user behaviours to service providers. Note that in the case where distribution of trust is not necessary, setting a trivial access structure (e.g., 1-out-of-1) recovers a simplified scheme that still benefits from proxy anonymity and revocation.

Example 1. A simple example of a policy is as follows: the user wants to prove they are aged 18 or above on a certain date. They would create the policy ψ as

$$\psi \coloneqq (\texttt{over\_18} : \texttt{T} \wedge \texttt{date} : \texttt{220626})$$

The Green box is an assertion being made by the user, whereas the Blue is a restriction on the proxy specifying when they can check this information. Together they form the policy; this should be interpreted as the user being at least 18 years of age which can be checked only on 22 June 2026. When the service provider receives this request, it computes its own context-dependent policy ψ' as

$$\psi \coloneqq (\texttt{over\_18} : \texttt{T} \wedge \texttt{date} : \texttt{220626})$$

Here, the Orange boxes are to be supplied by the service provider, enforcing validity of the asserted claims and within valid constraints. In particular, it checks the age claim against its own database and inserts T or F appropriately, and inserts the current date from a trusted time source. If either of these do not match ψ, then the scheme will reject the query, else ψ' will verify.

Remark 1. We observe that our construction can be used in applications wider than digital identity. In particular, we propose that it can be used to allow delegated access to encrypted files or disks. To realise this, the key commitment com would be used to seed a key derivation function, the key from which encrypts the

file or disk. Our PATS schemes would allow delegated recovery of the KDF seed, and thus of the encryption key. This would make our scheme PAT-I suitable for use in time-based delegation of full-disk encryption keys (e.g., LUKS[1]), requiring a trusted source of time (obtained from e.g., an onboard real-time clock).

Furthermore, PAT could be integrated into online authentication to permit (possibly time-bound) access to user accounts. This motivates distributed tokens as users may want to allow for secondary access but only under agreement of multiple trusted proxies.

2.1 Syntax for Policy-Based Access Tokens

In this section, we provide the syntax and correctness properties for both variants of the Policy-Based Access Tokens scheme. The first, PAT-I, in Definitions 1 and 2, and the second, PAT-II, in Definitions 3 and 4.

Definition 1 (PAT-I syntax). *A PAT-I scheme comprises four algorithms, permitting policy-based tokens to be generated from long-term shared secret, s.*

$\mathsf{Setup}(\lambda)$ *This algorithm takes as input a security parameter, λ, generates a description of the scheme and outputs a trapdoor* td *as part of public parameters* pp. Public parameters pp are implicitly input to all algorithms.

$\mathsf{UserKeyGen}(s)$ *This algorithm takes as input s from the key space $\mathcal{S}$, or $\perp$, and is run by the user to generate a key* key *and its commitment* com.

$\mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \psi)$ *Executed by the user, for some access policy ψ from the universe of policies Ψ, and key* key *with commitment* com*, it generates a token* tok.

$\mathsf{Verify}(\mathsf{tok}, \mathsf{com}, \psi')$ *This algorithm is run by the server. It verifies* tok *provided by the proxy, and outputs 1 if the token is valid for the key in* com*, and with respect to ψ'. Otherwise, it returns 0.*

Definition 2 (PAT-I correctness). *A PAT-I scheme is correct if, for all $\lambda \in \mathbb{N}$, $s \in \mathcal{S}$, $\psi \in \Psi$, and $\psi' \in \Psi$, the probability* $\Pr\left[\mathsf{Verify}(\mathsf{tok}, \mathsf{com}, \psi') = 1\right] = 1$ *given*

$$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda);\ (\mathsf{key}, \mathsf{com}) \leftarrow \mathsf{UserKeyGen}(s);\ \mathsf{tok} \leftarrow \mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \psi)$$

Definition 3 (PAT-II syntax). *A PAT-II scheme comprises seven algorithms, permitting distributed revocable policy-based tokens to be generated from long-term shared secret, s:*

$\mathsf{Setup}(\lambda)$ *This algorithm takes as input a security parameter λ, generates a description of the scheme and outputs a trapdoor* td *as part of public parameters* pp. Public parameters pp are implicitly input to all algorithms.

$\mathsf{UserKeyGen}(s)$ *This algorithm takes as input s from the key space $\mathcal{S}$, or $\perp$, and is run by the user to generate a key* key *and its commitment* com.

[1] `https://gitlab.com/cryptsetup/LUKS2-docs`

$\mathsf{Exp}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}I}}(\lambda)$

1 : $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
2 : $(\mathsf{key}, \mathsf{com}) \leftarrow \mathsf{UserKeyGen}(\bot)$
3 : $(\mathsf{tok}^\star, \psi^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Delegate}}}$
4 : **return** $\mathsf{Verify}(\mathsf{tok}, \psi, \mathsf{com}) \wedge \mathsf{tok}^\star \notin \mathsf{TL}$

$\mathcal{O}_{\mathsf{Delegate}}(\mathsf{key})$ on input ψ

1 : $\mathsf{tok} \leftarrow \mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \psi)$
2 : $\mathsf{TL} \leftarrow \mathsf{TL} \cup \{\mathsf{tok}\}$
3 : **return** tok

Figure 3: Token unforgeability experiment for PAT-I.

$\mathsf{ProxyKeyGen}()$ *This algorithm is executed by the proxy. When called, it generates a proxy key pair as* $(\mathsf{psk}, \mathsf{ppk})$.

$\mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \mathsf{ppk}_i, \psi)$ *Executed by the user, for some access policy* ψ *from the universe of policies* Ψ*, a set of* n *proxy public keys* PPK*, and a key* key *with commitment* com*, it returns info for each user* i*,* $\mathsf{INFO} \coloneqq \{\mathsf{info}_i\}_{i=1}^n$.

$\mathsf{Revoke}(\mathsf{dsk}, \mathsf{info}_i)$ *This algorithm is executed by the user to revoke the delegation access given to a proxy* i*. The user updates the revocation list* RL.

$\mathsf{Show}(\mathsf{psk}_i, \mathsf{info}_i)$ *Executed by a proxy* i*, with delegation information* info_i *and proxy secret key* psk_i*, this algorithm outputs a token* tok_i.

$\mathsf{Verify}(\mathsf{TOK}, \mathsf{com}, \mathsf{dpk}, \psi')$ *Run by the server, it checks that each* $\mathsf{tok} \in \mathsf{TOK}$ *is not in the revocation token list* RL*, it verifies if* TOK *is valid for the key in* com*, and with respect to* ψ'*. If these checks pass then it outputs 1, else 0.*

Definition 4 (PAT-II correctness). *A PAT-II scheme is correct if, for all* $\lambda \in \mathbb{N}$*,* $s \in \mathcal{S}$*,* $\psi \in \Psi$*,* $\psi' \in \Psi$*, the probability* $\Pr\left[\mathsf{Verify}(\mathsf{TOK}, \mathsf{com}, \mathsf{dpk}', \psi') = 1\right] = 1$ *given* $\forall i \in [1, n]$

$$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda);\ (\mathsf{key}, \mathsf{com}) \leftarrow \mathsf{UserKeyGen}(s);\ (\mathsf{psk}_i, \mathsf{ppk}_i) \leftarrow \mathsf{ProxyKeyGen}();$$
$$\mathsf{info}_i \leftarrow \mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \mathsf{ppk}_i, \psi);\ \mathsf{tok}_i \leftarrow \mathsf{Show}(\mathsf{psk}_i, \mathsf{info}_i)$$

2.2 Security of PAT-I

We define the *token unforgeability* security property for our Policy-Based Access Tokens scheme to capture a malicious proxy's ability to create tokens that successfully verify, but were not created by the user. This can be seen in the experiment defined in Figure 3; the experiment creates a set of public parameters, defines a user by running UserKeyGen to create challenge key $\mathsf{key}^\star$ with commitment $\mathsf{com}^\star$. An adversary is then tasked to create a token $\mathsf{tok}^\star$ that passes the verify algorithm. During this phase, it has access to a delegation oracle that creates valid tokens for $\mathsf{key}^\star$. To prevent the adversary from trivially winning, any tokens obtained from the oracle are not considered valid forgeries; this is enforced by the win condition on line 4 of the experiment using the list TokList. We note that we do not consider corruption of a server in this threat model due to the expected use case of the protocol, in which the service provider is trusted by the user. If the server were malicious or corrupt, then there would be no need to test tokens and instead allow direct adversarial access.

Definition 5 (Token Unforgeability). *A PAT-I scheme offers* Token Unforgeability *if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}I}}(\lambda) = 1\right] \leqslant \nu(\lambda).$$

The token unforgeability experiment is defined in Figure 3.

2.3 Security of PAT-II

We introduce an additional privacy property, *proxy anonymity*, for our PAT-II scheme that meets the requirement presented in Figure 1. It captures a fully malicious server's ability to link tokens across proxies, i.e., token unlinkability. The experiment is formally defined in Figure 4. It creates public parameters and a user key key with commitment com. The adversary generates two sets of proxies and a policy ψ. The set of proxy keys SL_b is used to create the set INFO_b via the delegate algorithm, which is passed to the adversary.

The adversary then is required to output its guess b', and it wins the experiment if $b' = b$. During the adversarial phase, we allow $\mathcal{A}$ to interact with the scheme using oracles. It uses the oracle $\mathcal{O}_{\mathsf{Add}}$ to create proxies in the scheme and $\mathcal{O}_{\mathsf{Corrupt}}$ to corrupt them, i.e., to obtain their secret key psk. To delegate tokens to a proxy set of its choosing it calls $\mathcal{O}_{\mathsf{Delegate}}$ on an input SL comprising a set of public keys ppk_i for $i \in [1, n]$. The adversary can also use the oracle $\mathcal{O}_{\mathsf{Revoke}}$ to revoke info_i. There are two additional checks in the win condition of the experiment in line 9, which ensures that any of the challenge keys have not been revoked nor corrupted, as this would allow the adversary to trivially win the game. Similarly, the challenge proxy sets are of equal size and both are valid for the policy ψ, which we write as $\psi(\mathsf{SL}) = 1$, otherwise $\psi(\mathsf{SL}) = 0$.

Definition 6 (Proxy Anonymity). *A PAT-II scheme offers* Proxy Anonymity *if, for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that*

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{ProxyAnon}-0}_{\mathcal{A},\mathsf{PAT\text{-}II}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{ProxyAnon}-1}_{\mathcal{A},\mathsf{PAT\text{-}II}}(\lambda) = 1\right]\right| \leqslant \nu(\lambda).$$

The proxy anonymity experiment is defined in Figure 4.

Modelled on PAT-I, *token unforgeability* for PAT-II is extended to allow for multiple proxies and a scheme that supports revocation. Precisely, it captures a malicious proxy's ability to create tokens that successfully verify, but which were not generated by a valid set of users. As with proxy anonymity, we allow $\mathcal{A}$ to interact with the scheme using $\mathcal{O}_{\mathsf{Add}}$, $\mathcal{O}_{\mathsf{Corrupt}}$, $\mathcal{O}_{\mathsf{Delegate}}$, and $\mathcal{O}_{\mathsf{Revoke}}$. The delegation oracle additionally tracks the token and public key pairs $(\mathsf{tok}^\star, \mathsf{ppk}^\star)$ that $\mathcal{A}$ queried. This enables the check on line 10 that ensures at least one token was not created by the delegation oracle, nor by a corrupt proxy. This also means we achieve a strong notion of unforgeability; the adversary can query the challenge proxies in $\mathsf{PPK}^\star$, winning provided it generates a *new* token $\mathsf{tok}^\star$ for some policy $\psi^\star$.

$\mathsf{Exp}^{\mathsf{ProxyAnon}-b}_{\mathcal{A},\mathsf{PAT\text{-}II}}(\lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{key}, \mathsf{com}, \mathsf{dsk}, \mathsf{dpk}) \leftarrow \mathsf{UserKeyGen}()$
$(\mathsf{psk}_0, \mathsf{ppk}_0) \leftarrow \mathsf{ProxyKeyGen}()$
$(\mathsf{psk}_1, \mathsf{ppk}_1) \leftarrow \mathsf{ProxyKeyGen}()$
$(\mathsf{SL}_0, \mathsf{SL}_1, \psi) \leftarrow \mathcal{A}_0^{\mathcal{O}_{\mathsf{Delegate}}, \mathcal{O}_{\mathsf{Revoke}}, \mathcal{O}_{\mathsf{Add}}, \mathcal{O}_{\mathsf{Corrupt}}}(\mathsf{key}, \mathsf{com}, \mathsf{dpk})$
$b \xleftarrow{\$} \{0, 1\}$
$(\mathsf{INFO}_b, \mathsf{SL}'_b) \leftarrow \mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \mathsf{SL}_b, \psi)$
$\mathsf{TOK}_b \leftarrow \mathsf{Show}(\mathsf{INFO}_b, \mathsf{SL}_b)$
$b' \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{Delegate}}, \mathcal{O}_{\mathsf{Revoke}}, \mathcal{O}_{\mathsf{Add}}, \mathcal{O}_{\mathsf{Corrupt}}}(\mathsf{TOK}_b)$
return $b' \stackrel{?}{=} b \wedge (\forall \mathsf{ppk}_i \in \{\mathsf{SL}_0, \mathsf{SL}_1\} : \mathsf{ppk}_i \notin \mathsf{RL} \cup \mathsf{CL})$
$\wedge\, |\mathsf{SL}_0| = |\mathsf{SL}_1| \wedge \psi(\mathsf{SL}_0) = \psi(\mathsf{SL}_1) = 1$

$\mathsf{Exp}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}II}}(\lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{com}, \mathsf{key}, \mathsf{dsk}, \mathsf{dpk}) \leftarrow \mathsf{UserKeyGen}()$
$\mathsf{TOK}^\star, \psi' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Delegate}}, \mathcal{O}_{\mathsf{Revoke}}, \mathcal{O}_{\mathsf{Add}}, \mathcal{O}_{\mathsf{Corrupt}}}$
return $\mathsf{Verify}(\mathsf{TOK}^\star, \mathsf{com}, \mathsf{dpk}, \psi')$
$\wedge \Big(\exists i \in [1, n] \text{ s.t. } \big(\mathsf{ppk}_i^\star \notin \mathsf{CL}\big) \wedge \big(\forall \mathsf{tok}_j \in \mathsf{TOK}^\star : (\mathsf{tok}_j, \mathsf{ppk}_i^\star) \notin \mathsf{TL}\big)\Big)$

$\mathcal{O}_{\mathsf{Delegate}}(\mathsf{key}, \mathsf{com}, \mathsf{psk}, \mathsf{ppk})$ on input ψ

$\mathsf{info} \leftarrow \mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \mathsf{ppk}, \psi)$
$\mathsf{tok} \leftarrow \mathsf{Show}(\mathsf{psk}, \mathsf{info})$
$\mathsf{TL} \leftarrow \mathsf{TL} \cup \{\mathsf{tok}, \mathsf{ppk}\}$
return tok

$\mathcal{O}_{\mathsf{Corrupt}}$ on input ppk

$\mathsf{psk} \leftarrow \mathsf{PL}(\mathsf{ppk})$
$\mathsf{CL} \leftarrow \mathsf{CL} \cup \{\mathsf{ppk}\}$
return psk

$\mathcal{O}_{\mathsf{Add}}$ when called

$(\mathsf{psk}, \mathsf{ppk}) \leftarrow \mathsf{ProxyKeyGen}$
$\mathsf{PL}(\mathsf{ppk}) \leftarrow \mathsf{psk}$
return ppk

$\mathcal{O}_{\mathsf{Revoke}}$ on input info

$\mathsf{RL} \leftarrow \mathsf{Revoke}(\mathsf{info})$
return RL

Figure 4: Proxy anonymity and token unforgeability experiments for PAT-II. The token list, TL, inside the dotted box, is only used for token unforgeability.

Definition 7 (Token Unforgeability). *A PAT-II offers* Token Unforgeability *if, for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}II}}(\lambda) = 1\right] \leqslant \nu(\lambda).$$

The token unforgeability experiment is defined in Figure 4.

3 Building Blocks

In this section, we formally present the syntax and security definitions of the building blocks we will use to build our Policy-Based Access Tokens scheme.

3.1 Chameleon Hash Functions

Chameleon (or trapdoor) hash functions were introduced by Krawczyk and Rabin [28] as a non-standard type of collision-resistant hash function associated with a private key, also known as a trapdoor, that enables the trapdoor holder to compute collisions efficiently. The formal Chameleon hash framework used in our approach is based upon the adaptation by Camenisch *et al.* [5].

Definition 8 (Chameleon Hash Function [5]). *A Chameleon hash function* CH *consists of five algorithms:*

$\mathsf{CParGen}(\lambda)$ *takes as input the security parameter λ and outputs public parameters of the scheme.* These public parameters pp_{ch} are implicitly given as input for all algorithms.

$\mathsf{CKGen}(\mathsf{pp_{ch}})$ *generates the key pair* $(\mathsf{sk}, \mathsf{pk})$ *of the scheme.*

$\mathsf{CHash}(\mathsf{pk_{ch}}, m)$ *takes public key* $\mathsf{pk_{ch}}$ *and a message m to hash, returning a hash h and some randomness r.*

$\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h)$ *is a deterministic algorithm that gets the public key* $\mathsf{pk_{ch}}$*, a message m, randomness r and a hash h as the input. It returns a decision $d \in \{0, 1\}$ indicating whether the hash h is valid.*

$\mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r, h)$ *on input of secret key* $\mathsf{sk_{ch}}$*, the old message m, the randomness r, hash h and a new message m', returns a new randomness r'.*

Chameleon hash correctness. A CH scheme is correct if, for all security parameters $\lambda \in \mathbb{N}$, public parameters $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(\lambda)$, key pairs $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$, message $m \in \mathcal{M}$, hash and randomness $(h, r) \leftarrow \mathsf{CHash}(\mathsf{pk_{ch}}, m)$, modified message $m' \in \mathcal{M}$, and adapted randomness $r' \leftarrow \mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r, h)$, the following holds:

$$\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h) = \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m', r', h) = 1$$

$\mathsf{Exp}^{\mathsf{CollRes}}_{\mathcal{A},\mathsf{CHash}}(\lambda)$

1 : $\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CParGen}(\lambda)$
2 : $(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) \leftarrow \mathsf{CKGen}()$
3 : $Q \leftarrow \emptyset$
4 : $(m^\star, r^\star, m'^\star, r'^\star, h^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Adapt}'}}(\mathsf{pk}_{\mathsf{ch}})$
5 : **return** $m'^\star \notin Q \ \wedge \ m^\star \neq m'^\star \ \wedge$
6 : $\quad \mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, m^\star, r^\star, h^\star) = 1 \ \wedge$
7 : $\quad \mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, m'^\star, r'^\star, h^\star) = 1$

$\mathcal{O}_{\mathsf{Adapt}'}(\mathsf{sk}, m, m', r, h)$

1 : **if** $\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, m, r, h) \neq 1$
2 : $\quad$ **return** $\perp$
3 : $r' \leftarrow \mathsf{Adapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', r, h)$
4 : **if** $r' = \perp$, **return** $\perp$
5 : $Q \leftarrow Q \cup \{m, m'\}$
6 : **return** r'

Figure 5: Collision resistance property for Chameleon hashes.

Security properties. We note that there are additional properties defined for this construction, such as indistinguishability and uniqueness. However, in this work, we will focus on one particular security property of Chameleon hash functions, namely collision resistance. This property ensures that, even with access to an adapt oracle, an adversary $\mathcal{A}$ cannot find any collisions for message other than those queried to the adapt oracle.

Definition 9 (Collision Resistance). CH *offers* Collision Resistance *if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that:*

$$\Pr\left[\mathsf{Exp}^{\mathsf{CollRes}}_{\mathcal{A},\mathsf{CHash}}(\lambda) = 1\right] \leqslant \nu(\lambda).$$

The collision resistance experiment is defined in Figure 5.

3.2 Asynchronous Remote Key Generation

ARKG [16] allows for asynchronous key delegation. It was first used in the analysis of Yubico's backup authentication protocol for WebAuthn; it has since been used to enable anonymous delegation in the context of WebAuthn [18] and enabled strong privacy properties in group signatures [32].

Definition 10 (Asynchronous Remote Key Generation [16]). *An* ARKG *scheme consists of five algorithms,* Setup, KGen, DerivePK, DeriveSK, Check, *specified as follows.*

Setup(λ) *takes as input the security parameter λ and outputs public parameters of the scheme.* We assume that $\mathsf{pp}_{\mathsf{ARKG}}$ is implicit input to all other ARKG algorithms.
KGen() *generates the key pair* ()sk, pk) *of the scheme when called.*
DerivePK(pk, aux) *takes as input the public key* pk *and auxiliary data* aux *and returns a new public key* ppk *and derivation data* cred.
DeriveSK(sk, cred) *takes input a long-term secret key* sk *and derivation data* cred *and returns derived private key* psk *for corresponding* ppk.
Check(psk, ppk) *returns 1 if* (psk, ppk) *form a valid key pair, otherwise 0.*

$\mathsf{Exp}^{\mathsf{pku}}_{\mathcal{A},\mathsf{ARKG}}(\lambda)$

1 : $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(\lambda)$
2 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KeyGen}()$
3 : $b \leftarrow\!\!\$\ \{0,1\}$
4 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{pku}}}(\mathsf{pk})$
5 : **return** $b' = b$

$\mathsf{Exp}^{\mathsf{hwKS}}_{\mathcal{A},\mathsf{ARKG}}(\lambda)$

1 : $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(\lambda)$
2 : $Q_{\mathsf{pk}'}, Q_{\mathsf{sk}'} \leftarrow \emptyset$
3 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathsf{pp})$
4 : $(\mathsf{sk}^\star, \mathsf{pk}^\star, \mathsf{cred}^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{dpk}}, \mathcal{O}_{\mathsf{dsk}}}(\mathsf{pk})$
5 : $\mathsf{sk}' \leftarrow \mathsf{DeriveSK}(\mathsf{sk}, \mathsf{cred}^\star)$
6 : **return** $\mathsf{Check}(\mathsf{pp}, \mathsf{sk}^\star, \mathsf{pk}^\star)$
7 : $\quad \wedge\ \mathsf{Check}(\mathsf{pp}, \mathsf{sk}', \mathsf{pk}^\star)$
8 : $\quad \wedge\ \mathsf{cred}^\star \notin Q_{\mathsf{sk}'}$

$\mathcal{O}_{\mathsf{pku}}(\mathsf{sk}, \mathsf{pk})$ when called

1 : $(\mathsf{pk}_0, \mathsf{cred}) \leftarrow\!\!\$\ \mathsf{DerivePK}(\mathsf{pk})$
2 : $\mathsf{sk}_0 \leftarrow \mathsf{DeriveSK}(\mathsf{sk}, \mathsf{cred})$
3 : $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow\!\!\$\ \mathsf{KeyGen}()$
4 : **return** $(\mathsf{sk}_b, \mathsf{pk}_b, \mathsf{cred})$

$\mathcal{O}_{\mathsf{dpk}}(\mathsf{pk}, \mathsf{aux})$

1 : $(\mathsf{pk}', \mathsf{cred}) \leftarrow \mathsf{DerivePK}(\mathsf{pk}, \mathsf{aux})$
2 : $Q_{\mathsf{pk}'} \leftarrow Q_{\mathsf{pk}'} \cup \{(\mathsf{pk}', \mathsf{cred})\}$
3 : **return** $(Q_{\mathsf{pk}'}, \mathsf{pk}', \mathsf{cred})$

$\mathcal{O}_{\mathsf{dsk}}(\mathsf{sk}, \mathsf{cred})$

1 : $\mathsf{sk}' \leftarrow \mathsf{DeriveSK}(\mathsf{sk}, \mathsf{cred})$
2 : $Q_{\mathsf{sk}'} \leftarrow Q_{\mathsf{sk}'} \cup \{\mathsf{cred}\}$
3 : **return** $(Q_{\mathsf{sk}'}, \mathsf{sk}')$

Figure 6: Experiments for ARKG's unlinkability and honest-weak key secrecy properties.

Security properties. An ARKG scheme provides public-key unlinkability if $\mathcal{A}$ is not able to distinguish between derived public keys and uniformly-sampled public keys. Additionally, honest-weak private-key secrecy is provided if $\mathcal{A}$ is not able to derive a valid key pair $(\mathsf{sk}^\star, \mathsf{pk}^\star)$ and corresponding $\mathsf{cred}^\star$ for an initial public key pk. Other variants of the private-key secrecy properties are provided in the original paper.

Definition 11 (PK Unlinkability). *An* ARKG *scheme offers* PK Unlinkability *if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that:*

$$\Pr\left[\mathsf{Exp}^{\mathsf{pku}}_{\mathcal{A},\mathsf{ARKG}}(\lambda) = 1\right] \leqslant \nu(\lambda).$$

The public-key unlinkability experiment is defined in Figure 6.

Definition 12 (Honest-Weak Key Secrecy). ARKG *offers* Honest-Weak Key Secrecy *if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that:*

$$\Pr\left[\mathsf{Exp}^{\mathsf{hwKS}}_{\mathcal{A},\mathsf{ARKG}}(\lambda) = 1\right] \leqslant \nu(\lambda).$$

The honest-weak key secrecy experiment is defined in Figure 6.

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

1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
2 : $(m^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{pk})$
3 : **if** $\mathsf{Verify}(\mathsf{pk}, m^\star, \sigma^\star) = 1 \wedge m^\star \notin \mathsf{QueryList}$
4 : **then return** 1
5 : **else return** 0

$\mathcal{O}_{\mathsf{Sign}}(\mathsf{sk}, m)$

1 : $\mathsf{QueryList} \leftarrow \mathsf{QueryList} \cup \{m\}$
2 : **return** $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$

Figure 7: Unforgeability (EUF-CMA) for digital signature schemes.

3.3 Digital Signatures

A digital signature scheme enables a user to produce a message-specific value, called a signature, which can later be verified by any third party to ensure the authenticity and integrity of the message.

Definition 13 (Digital Signature Scheme [26]). *A digital signature scheme consists of three algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *such that:*

$\mathsf{KeyGen}(\lambda)$ *outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*, where* pk *is the public verification key and* sk *is the secret signing key.*
$\mathsf{Sign}(\mathsf{sk}, m)$ *takes as input the secret key* sk *and a message* $m \in \{0,1\}^\star$*, and outputs a signature* σ*.*
$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ *takes as input the public key* pk*, a message* $m \in \{0,1\}^\star$*, and a signature* σ*, and outputs a bit* $b \in \{0,1\}$*, where 1 indicates acceptance and 0 indicates rejection.*

DS correctness. For all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and $m \in \{0,1\}^\star$, we have:

$$\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$$

Security properties. In our construction, we utilise a standard digital scheme that guarantees *existential unforgeability under chosen-message attacks* (EUF-CMA). It captures the requirement that even an adversary who can obtain valid signatures on arbitrary messages of its choice should be unable to forge a signature on a new message that was not queried to the signing oracle.

Definition 14 (EUF-CMA Unforgeability). *A* DS *scheme offers the* EUF-CMA Unforgeability *property if, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\nu(\cdot)$ *such that:*

$$\Pr\left[\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathcal{A},\mathsf{DS}}(\lambda) = 1\right] \leqslant \nu(\lambda).$$

The EUF-CMA experiment is defined in Figure 7.

$\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{SS}}(\lambda)$

1 : $k \leftarrow\!\!\$\ K$
2 : $(\psi, \mathcal{J}) \leftarrow \mathcal{A}_0()$
3 : $\{s_i\}_{i=1}^n \leftarrow \mathsf{Share}(k, \psi)$
4 : $k' \leftarrow \mathcal{A}_1((i, s_i)_{i\in\mathcal{J}})$
5 : **if** $k = k' \wedge (\mathcal{J}) \neq 1$ **then return** 1
6 : **else return** 0

Figure 8: Privacy experiment for secret sharing schemes.

3.4 Secret Sharing Schemes

Secret sharing allows for a party to split a secret k into shares s_i according to some policy ψ. These shares can be combined to recover the secret k if and only if the set of shares $\mathcal{I}$ satisfy ψ, which we denote as $\psi(\mathcal{I}) = 1$. In the literature, the set $\mathcal{I}$ is called an authorised set. In our work, we allow ψ to be any monotone access structure, which includes threshold and conjunctive policies, and which we consider to be most appropriate for our work.

Definition 15 (Secret Sharing Schemes [29]). *A Secret Sharing Scheme* SS *consists of two algorithms:*

$\mathsf{Share}(k, \psi)$ *On input of a secret* k *and policy* ψ*, this algorithm outputs shares* $s_1, \ldots, s_d$.

$\mathsf{Recover}(\{i, s_i\}_{i=1}^t, \psi)$ *on input of the pairs* (i, s_i) *for all* $i \in [1, t]$ *and candidate policy* ψ*, it either outputs* k *or* $\perp$.

SS correctness. An SS scheme is correct if, for every qualified set $\mathcal{I}$, reconstruct recovers the original secret k. That is, for any ψ, and any $\mathcal{I}$ such that $\psi(\mathcal{I}) = 1$, we have the following with all but negligible probability:

$$\{s_i\}_{i=1}^n \leftarrow \mathsf{Share}(k, \psi) \implies k \leftarrow \mathsf{Recover}(\{i, s_i\}_{i=1}^t, \psi)$$

Security properties. Security properties of these schemes require that k cannot be obtained by any corrupt set of shares that do not satisfy ψ.

Definition 16 (Privacy). *An* SS *scheme offers* Privacy *if, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\nu(\cdot)$ *such that:*

$$\Pr\left[\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{SS}}(\lambda) = 1\right] \leqslant \nu(\lambda).$$

The privacy experiment is defined in Figure 8.

$\mathsf{Setup}(\lambda)$

1 : $\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CH.ParGen}(\lambda)$
2 : $\mathsf{td}, \mathsf{pk}_{\mathsf{ch}} \leftarrow \mathsf{CH.CKGen}()$
3 : **return** $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ch}}, \mathsf{td}, \mathsf{pk}_{\mathsf{ch}})$

$\mathsf{UserKeyGen}(s)$

1 : **if** $s = \bot$, **then** $s \leftarrow\!\!\$\ \{0,1\}^{\lambda}$
2 : $(h, r) \leftarrow \mathsf{CH.Hash}(\mathsf{pp.pk}_{\mathsf{ch}}, s)$
3 : **return** $\mathsf{key} = (s, r)$, $\mathsf{com} = h$

$\mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \psi)$

1 : **parse** $\mathsf{key} = (s, r)$ and $\mathsf{com} = h$
2 : $r' \leftarrow \mathsf{CH.Adapt}(\mathsf{pp.td}, \psi, s, r, h)$
3 : **return** $\mathsf{tok} \leftarrow r'$

$\mathsf{Verify}(\mathsf{tok}, \mathsf{com}, \psi')$

1 : **parse** $\mathsf{com} = h$
2 : **if** $\mathsf{CH.CHashCheck}(\mathsf{pp.pk}_{\mathsf{ch}}, \psi', \mathsf{tok}, h)$
3 : **then return** 1, **else return** 0

Figure 9: Algorithms for our PAT-I construction.

4 Generic Construction of Policy-Based Access Tokens

We now present the constructions for PAT-I and PAT-II, based on the building blocks introduced in Section 3, and prove their security properties in this section.

4.1 PAT-I Construction

Here we detail the construction of PAT-I which is based on simplistic use of a Chameleon hash functions to achieve secure token delegation. To begin, the server executes the Setup algorithm to generate public parameters pp by running Chameleon hash CParGen algorithm followed by CKGen to generate the public key $\mathsf{pk}_{\mathsf{CHash}}$ and trapdoor td associated to a Chameleon hash function.

A user runs the UserKeyGen algorithm with input of their long-term key s. Using $\mathsf{pk}_{\mathsf{CHash}}$, it creates a hash h and randomness r by executing CHash. To delegate to a proxy, the user then runs the Delegate algorithm. This consists of executing CHash.Adapt with trapdoor td, policy ψ, hash h, long term key s, and randomness r as the input to produce delegation token $\mathsf{tok} = r'$. Intuitively, it creates a collision on hash h for the new message ψ.

The token is verified by the server by running the Verify algorithm. This entails running CH.CHashCheck on input of the user's hash h and the proxy's token $\mathsf{tok} = r'$. The server provides its instance of the policy ψ' from the policy description. If this check passes, and CH.CHashCheck passes, then the server outputs 1 and accepts the token, otherwise it returns 0. Intuitively, to pass verify, the proxy must have obtained a valid token, i.e., produced a collision on h for a message ψ' from the user. Otherwise, it would have broken collision resistance of the Chameleon hash CHash. The algorithms are presented in Figure 9.

Correctness. The correctness of the PAT-I construction presented in Figure 9 follows from the correctness of the building blocks DS and CHash.

Theorem 1. *PAT-I satisfies* Token Unforgeability *if the Chameleon hash function CH offers* Collision Resistance.

Proof. Game_0 is defined to be the experiment $\mathsf{Exp}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}I}}(\lambda)$.

Game_1 is defined as Game_0 but with the following changes: we introduce an additional check in line 5 of the game, namely, we check that $k \neq \psi^\star$. We observe that the difference in the advantage of the adversary winning Game_0 and Game_1 is bound by the probability that the adversary guesses s. Since $s \in \{0,1\}^\lambda$, the probability that this event occurs is $2^{-\lambda}$, and thus the difference in advantage for $\mathcal{A}$ between these two games is negligible.

We now examine the advantage an adversary $\mathcal{A}$ has against Game_1. In particular, we claim that any adversary that wins Game_1 can be used to be build an adversary $\mathcal{B}$ against the collision resistance property of CH. The adversary $\mathcal{B}$ is going invoke its experiment and receives some challenge parameters pp_{ch} and key $\mathsf{sk}_{\mathsf{CHash}}$. It passes public parameters to the token unforgeability adversary $\mathcal{A}$ and sets the trapdoor to be $\mathsf{td} = \mathsf{sk}_{\mathsf{CHash}}$. Adversary $\mathcal{B}$ must also simulate the delegate oracle to $\mathcal{A}$, however, it can run the delegate algorithm honestly as it has knowledge of (td, s, r, h) and $\mathcal{A}$ provides the input ψ.

After some time, the adversary $\mathcal{A}$ returns the tuple $(\mathsf{tok}^\star, \psi^\star)$. Then $\mathcal{B}$ builds its response to the CR game by constructing the tuple $(h, m = k, r, m' = \psi^\star, r' = \mathsf{tok}^\star)$. Then we observe that if $\mathcal{A}$ wins its token unforgeability game, then $\mathcal{B}$ wins its collision resistance game.

Thus, the sequence of games Game_0 and Game_1 we have shown that:

$$\mathsf{Adv}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}I}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{CollRes}}_{\mathcal{B},\mathsf{CHash}}(\lambda).$$

Hence we conclude that PAT-I satisfies *Token Unforgeability* if the Chameleon hash function CH offers *Collision Resistance*. □

4.2 PAT-II Construction

This section details the construction of PAT-II, which is constructed from a Chameleon hash function CH, a digital signature DS, an Asynchronous Remote Key Generation scheme ARKG, and a secret sharing scheme SS.

As before, the server creates the public parameters pp of the scheme by running the Setup algorithm. In particular, it executes $\mathsf{CHash.CParGen}$, followed by the CKGen algorithm to generate the public key $\mathsf{pk}_{\mathsf{CHash}}$ and trapdoor td associated to a Chameleon hash function, then calls the setup for ARKG and DS. A user runs the $\mathsf{UserKeyGen}$ algorithm with their long term key s as input. Using $\mathsf{pk}_{\mathsf{CHash}}$, it creates a hash h and randomness r by executing CHash. Furthermore, it creates a revocation key pair $(\mathsf{dsk}, \mathsf{dpk}) \leftarrow \mathsf{DS.KGen}()$.

In PAT-II, proxies also need key pairs $(\mathsf{psk}, \mathsf{ppk})$. This is generated in the $\mathsf{ProxyKeyGen}$ algorithm by running $\mathsf{ARKG.KGen}$. In order to delegate to a (set of) proxy(ies), the user then runs the $\mathsf{Delegate}$ algorithm. It creates a derived public key ppk' on the proxy's public key ppk to produce delegation credentials cred. Then, as before, this comprises executing $\mathsf{CHash.Adapt}$ with trapdoor td, policy ψ, hash h, long term key s, and randomness r as the input to produce delegation token $\mathsf{tok} = r'$. In this step, the policy ψ includes the proxy's derived public key ppk' to bind the collision randomness r' to the proxy. Then, r' is split

$\mathsf{Setup}(\lambda)$

$\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CH.ParGen}(\lambda)$
$\mathsf{td}, \mathsf{pk}_{\mathsf{ch}} \leftarrow \mathsf{CH.CKGen}()$
return $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ch}}, \mathsf{td}, \mathsf{pk}_{\mathsf{ch}})$

$\mathsf{UserKeyGen}(s)$

if $s = \perp$, **then** $s \leftarrow\!\!\$\ \{0,1\}^{\lambda}$
$(h, r) \leftarrow \mathsf{CH.Hash}(\mathsf{pp.pk}_{\mathsf{ch}}, s)$
$(\mathsf{dsk}, \mathsf{dpk}) \leftarrow \mathsf{DS.KeyGen}()$
return $\mathsf{key} = (s, r), \mathsf{com} = h,$ $\mathsf{dsk}, \mathsf{dpk}$

$\mathsf{ProxyKeyGen}(\mathsf{pp})$

$(\mathsf{psk}, \mathsf{ppk}) \leftarrow \mathsf{ARKG.KeyGen}()$
return $\mathsf{psk}, \mathsf{ppk}$

$\mathsf{Revoke}(\mathsf{dsk}, \mathsf{info})$

parse ppk' from info
$\mathsf{RL} \leftarrow \mathsf{RL} \cup \{\mathsf{ppk}'\}$
$\sigma_{RL} \leftarrow \mathsf{Sign}(\mathsf{dsk}, \mathsf{RL})$
return RL, σ_{RL}

$\mathsf{Delegate}(\mathsf{key}, \mathsf{com}, \mathsf{PPK}, \psi)$

parse $\mathsf{key} = (s, r)$ and $\mathsf{com} = h$
$(\mathsf{PPK}', \mathsf{CRED}) \leftarrow \mathsf{ARKG.DerivePK}(\mathsf{PPK})$
$r' \leftarrow \mathsf{CH.Adapt}(\mathsf{pp.td}, \psi, h, s, r)$
$\{r'_i\} \leftarrow \mathsf{SS.Share}(r', \psi)$
return $\mathsf{info}_i = (r'_i, \mathsf{cred}_i, \mathsf{ppk}'_i) \forall i \in [1:n]$

$\mathsf{Show}(\mathsf{psk}_i, \mathsf{info}_i)$

parse $\mathsf{info}_i = (r'_i, \mathsf{cred}_i, \mathsf{ppk}'_i)$
$\mathsf{psk}' \leftarrow \mathsf{ARKG.DeriveSK}(\mathsf{psk}, \mathsf{cred})$
if $\mathsf{psk}'_i = \perp$ **then abort**
$\pi_i \leftarrow \mathsf{DS.Sign}(\mathsf{psk}'_i, r'_i)$
return $\mathsf{tok} = (r'_i, \pi_i, \mathsf{ppk}'_i)$

$\mathsf{Verify}(\mathsf{TOK}, \mathsf{com}, \mathsf{dpk}, \psi')$

parse $\mathsf{tok}_i = (r'_i, \pi_i\, \mathsf{ppk}_i)$ and $\mathsf{com} = h$
$r' \leftarrow \mathsf{SS.Recover}(\mathsf{TOK}, \psi)$
if $\mathsf{CH.CHashCheck}(\mathsf{pp.pk}_{\mathsf{ch}}, \psi', r', h)$
  $\wedge\ \mathsf{ppk}' \notin \mathsf{RL}$ for each $i \in [1, n]$
  $\wedge\ \mathsf{DS.Verify}(\pi_{\mathsf{i}}, \mathsf{ppk}'_{\mathsf{i}})$ for each $i \in [1, n]$
  $\wedge\ \mathsf{DS.Verify}(\sigma_{RL}, \mathsf{dpk})$
  then return 1, **else return** 0

Figure 10: Algorithms for our PAT-II construction.

by the secret sharing scheme SS into n shares r'_i according to the policy ψ. To revoke a proxy, the user appends ppk' to the revoke list RL and signs it using revocation key dsk.

A proxy processes its token tok using the Show algorithm to randomise their public key, yet demonstrates that it is the intended recipient of the token. To do this, it signs the tok with its derived secret key psk' (corresponding to ppk') computed from the DeriveSK algorithm, to create the signature π. Intuitively, this randomisation is what enables proxy anonymity.

A set of proxies present a set of tokens TOK (containing keyset PPK) to be verified by the server, which runs the Verify algorithm. It reconstructs r' from the proxies' tokens $\mathsf{tok}_i = r'_i$, by running the Recover algorithm of the SS scheme. The verifier ensures that, on input of the user's hash h, CH.CHashCheck verifies against the server-provided policy ψ'. Furthermore, it validates the authenticity of the revocation list RL (using dpk) and also checks that $\mathsf{ppk}' \notin \mathsf{RL}$ for each ppk' in PPK, and validates each token by verifying the signature π_i. If all checks pass, then the server outputs 1 and accepts the token, otherwise it returns 0.

Intuitively, the proxy must have obtained a valid token, otherwise it would have broken collision resistance of the Chameleon hash CHash, broken privacy of the secret sharing scheme SS, forged a signature π for a derived public key of a different proxy, or been able to recover psk' from a ppk', breaking honest-weak security of ARKG. A formal description of the algorithms is presented in Figure 10.

Remark 2. We note PAT-II can support fine-grained delegation control through revocation of a single proxy by setting the proxy set to be size 1 and omitting the secret sharing step. However, we include it in our scheme for more generality.

Correctness. The correctness of the PAT-II construction follows from the correctness of the building blocks DS, CHash, SS, and ARKG.

Theorem 2. *A PAT-II scheme satisfies* Proxy Anonymity *if* ARKG *offers* PK Unlinkability.

Proof. Let $\mathsf{Game}_0^{(b)}$ be defined as $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ProxyAnon}\text{-}b}(\lambda)$, indexed on b.

Let $\mathsf{Game}_1^{(b)}$ be defined as $\mathsf{Game}_0^{(0)}$, i.e., we fix the challenge bit $b = 0$. We argue that the advantage of $\mathcal{A}$ distinguishing between $\mathsf{Game}_1^{(b)}$ and $\mathsf{Game}_0^{(b)}$ is bound by an adversary $\mathcal{B}$ against the unlinkability property of ARKG.

To see this, consider line 6 in both games, and notice that the only input in the $\mathsf{Delegate}$ algorithm affected by the choice of b is ppk_b. The function $\mathsf{CH.Adapt}$ in $\mathsf{Delegate}$, which provides tok_b, uses the derived public key ppk'_b, derived from public key ppk_b via $\mathsf{ARKG.DerivePK}(\mathsf{ppk_b}, \mathsf{aux})$. If ppk'_b is unlinkable from ppk_b, tok_b is also unlinkable from ppk_b. Note the secret sharing scheme SS only depends on key, which is fixed; thus does not impact the adversary's ability to guess b.

We introduce an adversary $\mathcal{B}$ against the unlinkability property of ARKG. $\mathcal{B}$ invokes its experiment and creates the challenge for $\mathcal{A}$ with $\mathsf{pk}_0 = \mathsf{ppk}_b$ and the fixed inputs $(\mathsf{td}, \psi, \mathsf{ppk}, s, r, h)$. After some time, the adversary $\mathcal{A}$ outputs $(\mathsf{tok}_b, \mathsf{cred}_b, \mathsf{ppk}'_b)$. $\mathcal{A}$ outputs its guess at the challenge bit b, which $\mathcal{B}$ forwards as its response in the unlinkability experiment. $\mathcal{B}$ wins if $\mathcal{A}$ also wins. Thus:

$$\left|\Pr\left[\mathsf{Game}_1^{(1)} = 1\right] - \Pr\left[\mathsf{Game}_0^{(1)} = 1\right]\right| \leqslant \mathsf{Adv}^{\mathsf{pku}}_{\mathcal{B},\mathsf{ARKG}}(\lambda)$$

We observe that the advantage of the adversary distinguishing $\mathsf{Game}_1^{(0)}$ and $\mathsf{Game}_1^{(1)}$ is necessarily $\frac{1}{2}$ as it is independent of the bit b. Similarly, $\mathsf{Game}_0^{(0)}$ and $\mathsf{Game}_1^{(0)}$ are also indistinguishable as there is no change. Finally, we have shown that any adversary that can distinguish $\mathsf{Game}_0^{(1)}$ and $\mathsf{Game}_1^{(1)}$ can be used to build an adversary $\mathcal{B}$ against the unlinkability property of ARKG. Thus, by the triangle inequality, we conclude:

$$\mathsf{Adv}^{\mathsf{ProxyAnon}}_{\mathcal{A},\mathsf{PAT\text{-}II}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{pku}}_{\mathcal{B},\mathsf{ARKG}}(\lambda)$$

Thus PAT-II satisfies *Proxy Anonymity* if ARKG offers *PK Unlinkability.* □

Theorem 3. *A PAT-II scheme satisfies* Token Unforgeability *if* ARKG *offers* Honest-Weak Key Secrecy*,* CHash *offers* Collision Resistance*,* DS *offers* EUF-CMA Unforgeability *and* SS *offers* Privacy*.*

Proof. We observe that the win condition requires that a token provided by the adversary passes verification and was not obtained through the delegation oracle. Passing verification requires 4 checks: (1) $\mathsf{CH.CHashCheck}(\mathsf{td}, \psi', \mathsf{ppk}', r', h)$, (2) $\mathsf{DS.Verify}(\pi, \mathsf{ppk}')$, (3) $\mathsf{DS.Verify}(\sigma_{RL}, \mathsf{dpk})$, and (4) $\mathsf{ppk}' \notin \mathsf{RL}$. The adversary $\mathcal{A}$ could attack any of the checks 1–3 in order to create a forged token, whereas check 4 prevents a trivial win.

Denote E_i as the adversary winning the experiment by attack check (i). We now consider the reduction for E_1, and split further into two subgames conditioned on whether the shares contained in TOK successfully recover tok and were created by the game. Thus, let E_1^a be the case that $\mathsf{tok}^* = \mathsf{tok}$, and let E_1^b be the case $\mathsf{tok}^* \neq \mathsf{tok}$.

In E_1^a, the adversary returns a set of shares that create the delegated collision tok. However, the game prevents the adversary from corrupting all proxies; there must be at least one honest proxy. To set up this reduction, the adversary $\mathcal{B}$ invokes its experiment against privacy of the secret sharing scheme. It challenges $\mathcal{A}$ against E_1^a and waits for it to output a policy ψ, which may contain many additional constraints, but will also describe an access structure for the secret sharing policy, denoted $\tilde{\psi}$. The adversary $\mathcal{B}$ forwards $\tilde{\psi}$ to the privacy game. Next, it is issued with a set of shares $\{i, s_i\}_{i \in \mathcal{J}}$ for some maximal non-accepting set $\mathcal{J}$. When $\mathcal{A}$ calls the delegate oracle on a proxy belonging to $\mathcal{J}$, these shares are embedded as $\mathsf{tok}_i^\star$. Eventually, $\mathcal{A}$ outputs a set of tokens TOK. $\mathcal{B}$ runs the $\mathsf{Recover}$ algorithm to produce a candidate secret k', which it forwards to the privacy experiment. It wins the experiment if $k = k'$. Since $\mathcal{J}$ is non-accepting, any time $\mathcal{A}$ wins E_1^a, $\mathcal{B}$ wins against privacy of the secret sharing scheme.

We observe the experiment E_1^b follows the argument of Theorem 1, so we omit this for brevity. The conclusion is that any adversary able to win game E_1^b can be used to create an adversary $\mathcal{C}$ against collision resistance of CHash, since it provided tokens that produced a distinct but valid collision. Thus, we conclude:

$$\Pr[\mathsf{E}_1 = 1] \leqslant \mathsf{Adv}^{\mathsf{Priv}}_{\mathcal{B},\mathsf{SS}}(\lambda) + \mathsf{Adv}^{\mathsf{CollRes}}_{\mathcal{C},\mathsf{CHash}}(\lambda)$$

We now consider E_2. Game_0 is defined to be the experiment $\mathsf{Exp}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}II}}$ (conditioned on the event E_2). Game_1 is defined as Game_0 but with the following change: we change the requirement in line 4 of the game from the adversary providing $\mathsf{psk}^\star$ instead of $\psi^\star$. We observe that the difference in the advantage of the adversary winning Game_0 and Game_1 is bound by the probability that the adversary is able to create a signature $\pi^\star$ without knowledge of the secret key $\mathsf{psk}^\star$, i.e., the EUF-CMA property. $\mathcal{B}$ embeds the challenge key $\mathsf{pk}^\star$ as $\mathsf{ppk}^\star$, and it sets $\mathsf{psk}^\star = \perp$. We note that $\mathcal{B}$ can simulate the delegate oracle for the target $\mathsf{ppk}^\star$ by forwarding queries to its signing oracle. If $\mathcal{A}$ wins, then $\{\mathsf{tok}^\star, \mathsf{ppk}^\star\} \notin \mathsf{TL}$, which means that it did not query the delegate oracle for the token. Thus, any

signature $\pi^\star$ is a valid forgery and wins the EUF-CMA game. Therefore, we have:

$$\Pr[\mathsf{Game}_0 - \mathsf{Game}_1] \leqslant \mathsf{Adv}^{\mathsf{euf-cma}}_{\mathcal{D},\mathsf{DS}}(\lambda)$$

We claim that any adversary that wins Game_1 can be used to be build an adversary $\mathcal{B}$ against the honest-weak key secrecy property of ARKG. Adversary $\mathcal{B}$ invokes its experiment and receives challenge parameters $\mathsf{sk}_{\mathsf{psk}}$ and key $\mathsf{pk}_{\mathsf{ppk}}$. It passes public parameters to the token unforgeability adversary $\mathcal{A}$ and sets the proxy's public key to be $\mathsf{pk} = \mathsf{ppk}$. Adversary $\mathcal{B}$ must also simulate the delegate oracle to $\mathcal{A}$, for which it uses $\mathcal{O}_{\mathsf{ppk}^\star}$. After some time, the adversary $\mathcal{A}$ returns the tuple $(\mathsf{tok}^\star, \mathsf{psk})$. Then $\mathcal{E}$ builds its response to the key secrecy game by deriving psk' using the psk provided by $\mathcal{A}$. We observe that if $\mathcal{A}$ provides a correct psk', then $\mathcal{E}$ wins its key secrecy game. Thus:

$$\Pr[\mathsf{E}_2 = 1] \leqslant \mathsf{Adv}^{\mathsf{hwKS}}_{\mathcal{E},\mathsf{ARKG}}(\lambda) + \mathsf{Adv}^{\mathsf{euf-cma}}_{\mathcal{D},\mathsf{DS}}(\lambda)$$

We now consider E_3. In particular, the adversary can attack the signature verification in line 1 of the verify algorithm, which verifies RL. Game_0 is defined to be the experiment $\mathsf{Exp}^{\mathsf{TokenUnforge}}_{\mathcal{A}}(\lambda)$ (conditioned on the event E_3).

Game_1 is defined as Game_0, except we change the requirement in line 4 of the game from the adversary providing σ_{RL} to dsk. We observe that the difference in the advantage of the adversary winning Game_0 and Game_1 is bound by the probability that the adversary is able to create a signature σ_{RL} without knowledge of the secret key dsk. This probability is bound by the EUF-CMA property of the digital signature. In particular, $\mathcal{D}$ embeds the challenge key as dpk, and it sets $\mathsf{dsk} = \perp$. We note that $\mathcal{D}$ can simulate the revoke oracle for the target dpk by forwarding queries to its signing oracle. If $\mathcal{A}$ wins, it is able to create a forgery σ_{RL}. Thus $\mathcal{D}$ wins the EUF-CMA game. Thus, we have

$$\Pr[\mathsf{E}_3 = 1] \leqslant \mathsf{Adv}^{\mathsf{euf-cma}}_{\mathcal{D},\mathsf{DS}}(\lambda)$$

which gives a total bound of

$$\mathsf{Adv}^{\mathsf{TokenUnforge}}_{\mathcal{A},\mathsf{PAT\text{-}II}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{Priv}}_{\mathcal{B},\mathsf{SS}}(\lambda) + \mathsf{Adv}^{\mathsf{CollRes}}_{\mathcal{C},\mathsf{CHash}}(\lambda) + 2\mathsf{Adv}^{\mathsf{euf-cma}}_{\mathcal{D},\mathsf{DS}}(\lambda) + \mathsf{Adv}^{\mathsf{hwKS}}_{\mathcal{E},\mathsf{ARKG}}(\lambda).$$

Hence, we conclude PAT-II satisfies *Token Unforgeability* if ARKG offers *Honest-Weak Key Secrecy*, CHash offers *Collision Resistance*, DS offers *EUF-CMA Unforgeability* and SS offers *Privacy*. □

5 Instantiation and Implementation

Our Policy-Based Access Tokens constructions are built from generic primitives, so we state here suitable primitives from which one can instantiate our scheme. For efficiency reasons, we opt for a group-based instantiation. For the Chameleon hash we use the implementation by Tian [41], for the DS scheme we use ECDSA [33] as a standardised primitive, and for ARKG, we use the generalised construction from Frymann *et al.* [16], which is compatible with pairing

Table 1: Timing and sizes, averaged over 100 runs on an Intel i5 CPU (2.7GHz), of our Python implementation. For PAT-II, we use one proxy with a trivial policy.

	Sizes (kB)			Timing (ms)		
Protocol	User key	Proxy key	Token	Delegate	Show	Verify
PAT-I	6.96	–	6.88	408	–	85.3
PAT-II	7.04	1.64	8.59	593	58.9	113

groups [19]. User secret keys comprise two elements in $2\lambda\texttt{bits}$ and, $2\lambda\texttt{bits} + \mathbb{Z}_p^\star$ (for $p = \Theta(2^\lambda)$), respectively. The user public key in PAT-II is in $\mathbb{G}$ (cyclic group of order n). We have tok, which has $\lambda\texttt{bits}$ and $2\lambda\texttt{bits} + 2\mathbb{Z}_n^\star + \mathbb{G}$, for PAT-I and PAT-II respectively. The size of tok is determined solely by the Chameleon hash parameters (i.e., the randomness space $\mathcal{R}$) and the parameters of the signature scheme, and is therefore independent of the size of the policy ψ. Hence, a more complex policy ψ will not increase the size of tok. In PAT-II, proxy keys $(\mathsf{psk}, \mathsf{ppk})$ belong to $\mathbb{Z}_n \times \mathbb{G}$. If we compare this to approaches based on anonymous credentials, which are much more costly, e.g., the state-of-the-art SAAC scheme by Chairattana-Apirom *et al.* [8] has a credential size of $3\mathbb{G} + 9\mathbb{Z}_n$ (for pairings) and $7\mathbb{G} + 15\mathbb{Z}_n$ (for DDH). SAAC also requires input from a trusted helper before the showing phase, only achieving security guarantees in the random oracle model.

By using verifier local revocation, our delegation time is constant in the number of delegations and revocations, but verification is linear in the size of the revocation list RL and with the number of proxies (N), i.e., it has order $\mathcal{O}(N \cdot |\mathsf{RL}|)$. However, since the revocation list RL is typically orders of magnitude larger than the number of proxies (N), $\mathcal{O}(|\mathsf{RL}|)$ is the dominant factor determining practical performance. Asymptotically, this matches many state-of-the-art anonymous credential schemes such as that of Connolly *et al.* [11], however, some schemes (e.g. [8]) could support logarithmic set membership proofs to achieve $\mathcal{O}(\sqrt{|\mathsf{RL}|})$ [2]. Table 1 details concrete performance metrics based on our reference implementation [34].

6 Conclusion

We introduced a new primitive, Policy-Based Access Tokens, enabling privacy-preserving digital ID schemes, comprising two variants. PAT-I offers token unforgeability against an adversary that can query tokens and PAT-II, which extends this to capture multiple proxies and introduces a revocation mechanism. We define a new property, *proxy anonymity*, capturing token unlinkability which affords the user privacy against the verifying server, thus preventing metadata surveillance. We gave generic constructions and proved they meet our security properties under standard assumptions in the standard model. Our schemes were instantiated in the group setting using a proof-of-concept implementation, writ-

ten in Python. Benchmarking demonstrates efficiency, with delegation taking 408 and 593ms, and verification 85.3 and 113ms, for PAT-I and PAT-II respectively.

Future Work. Given a recent focus from standardisation bodies to develop quantum-resistant cryptography, it would be desirable to consider post-quantum instantiations for Policy-Based Access Tokens as future work. We note that lattice-based Chameleon hash functions (e.g. [7, 30]) and ARKG [4, 17, 38] exist, therefore such an instantiation is feasible.

References

1. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable Signatures. In: Computer Security – ESORICS 2005, pp. 159–177. Springer Berlin Heidelberg (2005)
2. Benarroch, D., Campanelli, M., Fiore, D., Gurkan, K., Kolonelos, D.: Zero-knowledge proofs for set membership: efficient, succinct, modular. Des. Codes Cryptography **91**(11), 3457–3525 (2023). https://doi.org/10.1007/s10623-023-01245-1
3. Bootle, J., Lyubashevsky, V., Nguyen, N.K., Sorniotti, A.: A Framework for Practical Anonymous Credentials from Lattices. In: CRYPTO 2023. Springer Nature Switzerland (2023)
4. Brendel, J., Clermont, S., Fischlin, M.: Post-quantum Asynchronous Remote Key Generation for FIDO2. In: ASIACRYPT 2024, pp. 465–493. Springer-Verlag (2024). https://doi.org/10.1007/978-981-96-0891-1_15
5. Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Chameleon Hashes with Ephemeral Trapdoors. In: Public-Key Cryptography – PKC 2017. Springer Berlin Heidelberg (2017)
6. Camenisch, J., Lysyanskaya, A.: An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In: EUROCRYPT 2001. Springer Berlin Heidelberg (2001)
7. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai Trees, or How to Delegate a Lattice Basis. In: EUROCRYPT 2010. Springer (2010)
8. Chairattana-Apirom, R., Harding, F., Lysyanskaya, A., Tessaro, S.: Server-Aided Anonymous Credentials. In: Advances in Cryptology – CRYPTO 2025, pp. 291–324. Springer Nature Switzerland (2025)
9. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985). https://doi.org/10.1145/4372.4373
10. Choi, J., Jung, S.: A handover authentication using credentials based on chameleon hashing. Communications Letters, IEEE **14** (2010)
11. Connolly, A., Deschamps, J., Lafourcade, P., Perez Kempner, O.: Protego: Efficient, Revocable and Auditable Anonymous Credentials with Applications to Hyperledger Fabric. In: Progress in Cryptology – INDOCRYPT 2022: 23rd International Conference on Cryptology in India, Kolkata, India, December 11–14, 2022, Proceedings. Springer-Verlag (2022). https://doi.org/10.1007/978-3-031-22912-1_11
12. Debes, H.B., Giannetsos, T.: RETRACT: Expressive Designated Verifier Anonymous Credentials. In: Proceedings of the 18th International Conference on Availability, Reliability and Security. ARES '23. ACM (2023)

13. Desmoulins, N., Dumanois, A., Kane, S., Traoré, J.: Making BBS Anonymous Credentials eIDAS 2.0 Compliant, Cryptology ePrint Archive, Paper 2025/619 (2025). https://eprint.iacr.org/2025/619.
14. Disclosure and Barring Service, Check a tenant's right to rent in England: Use their share code, (2024). https://www.gov.uk/view-right-to-rent.
15. European Commission, European Digital Identity, https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity.
16. Frymann, N., Gardham, D., Kiefer, F., Lundberg, E., Manulis, M., Nilsson, D.: Asynchronous Remote Key Generation: An Analysis of Yubico's Proposal for W3C WebAuthn. In: ACM CCS 2020, pp. 939–954. ACM (2020)
17. Frymann, N., Gardham, D., Manulis, M.: Asynchronous Remote Key Generation for Post-Quantum Cryptosystems from Lattices. In: 8th IEEE European Symposium on Security and Privacy, EuroS&P 2023. IEEE (2023)
18. Frymann, N., Gardham, D., Manulis, M.: Unlinkable Delegation of WebAuthn Credentials. In: Computer Security – ESORICS 2022. Springer Nature Switzerland
19. Frymann, N., Gardham, D., Manulis, M., Nartz, H.: Generalised Asynchronous Remote Key Generation for Pairing-Based Cryptosystems. In: Applied Cryptography and Network Security, ACNS 2023. LNCS, Springer, Heidelberg (2023)
20. Government Digital Service, Disclosure & Barring Service, (2024). https://www.gov.uk/government/organisations/disclosure-and-barring-service.
21. Government Digital Service, Prove your right to work to an employer. Tech. rep., (2024)
22. Government Digital Service, UK digital identity and attributes trust framework: Gamma 0.4 (pre-release). In: Department for Science, Innovation and Technology (2025). https://www.gov.uk/government/publications/uk-digital-identity-and-attributes-trust-framework-04/uk-digital-identity-and-attributes-trust-framework-gamma-04-pre-release
23. Group, E.D.I.C.: Architecture and Reference Framework (ARF), (2025). https://eudi.dev/latest/architecture-and-reference-framework-main/%5C#7435-risks-and-mitigation-measures-related-to-user-privacy
24. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated Verifier Proofs and Their Applications. In: EUROCRYPT. Springer Berlin Heidelberg (1996)
25. Kakvi, S.A., Martin, K.M., Putman, C., Quaglia, E.A.: SoK: Anonymous Credentials. In: Security Standardisation Research. Springer Nature Switzerland (2023)
26. Katz, J.: "Digital Signatures". In: Digital Signatures. Springer US, 2010. Chap. 1, pp. 9–10.
27. Kirupanithi, D.N., Antonidoss, A.: Self-Sovereign Identity Management System on blockchain based applications using Chameleon Hash. In: 2nd International Conference on Smart Electronics and Communication (ICOSEC) (2021)
28. Krawczyk, H., Rabin, T.: Chameleon Signatures. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA. The Internet Society (2000)
29. Krenn, S., Lorünser, T.: An Introduction to Secret Sharing: A Systematic Overview and Guide for Protocol Selection (2023)
30. Li, Y., Liu, S.: Tagged Chameleon Hash from Lattices and Application to Redactable Blockchain. In: Public-Key Cryptography – PKC 2024. Springer Nature Switzerland (2024)
31. Lin, H.-Y., Tsai, T.-T., Wu, H.-L.: A Revocable Certificateless Sanitizable Signature Scheme With Batch Verification. IEEE Access **12**, 143392–143400 (2024)

32. London, C., Gardham, D., Dragan, C.C.: Dynamic Group Signatures with Verifier-Local Revocation. In: 2025 IEEE 38th Computer Security Foundations Symposium (CSF), pp. 473–488. IEEE Computer Society (2025). https://doi.org/10.1109/CSF64896.2025.00026. https://doi.ieeecomputersociety.org/10.1109/CSF64896.2025.00026
33. National Institute of Standards and Technology, Digital Signature Standard (DSS). Tech. rep., (2013). https://doi.org/10.6028/nist.fips.186-4
34. Policy-based Access Tokens source code, https://gitlab.surrey.ac.uk/sccs/pat/.
35. Prime Minister's Office, New Digital ID scheme to be rolled out across UK, (2025). https://www.gov.uk/government/news/new-digital-id-scheme-to-be-rolled-out-across-uk.
36. Sabramooz, M.R., Zhang, K.: ChameleonRev: A Novel Approach to Efficient and Granular Control Credential Revocation on the Blockchain. In: 2024 6th International Conference on Blockchain Computing and Applications (BCCA) (2024)
37. Samelin, K., Slamanig, D.: Policy-Based Sanitizable Signatures. In: CT-RSA 2020, pp. 538–563. Springer International Publishing (2020)
38. Stebila, D., Wilson, S.: Quantum-Safe Account Recovery for WebAuthn. In: 19th Asia Conference on Computer and Communications Security, ASIA CCS. ACM (2024)
39. Steinfeld, R., Bull, L., Wang, H., Pieprzyk, J.: Universal Designated-Verifier Signatures. In: ASIACRYPT 2003. Springer Berlin Heidelberg (2003)
40. Tian, Y., Li, N., Li, Y., Szalachowski, P., Zhou, J.: Policy-based Chameleon Hash for Blockchain Rewriting with Black-box Accountability. In: 36th Annual Computer Security Applications Conference. ACSAC '20. ACM (2020)
41. Tian, Y.: Accountable Blockchain Rewriting. GitHub repository https://github.com/Yangguang-Tian-Surrey/Accountable-Blockchain-Rewriting (2020)
42. Verheul, E.R.: Self-Blindable Credential Certificates from the Weil Pairing. In: ASIACRYPT 2001. Springer Berlin Heidelberg (2001)
43. Xu, S., Ning, J., Ma, J., Xu, G., Yuan, J., Deng, R.H.: Revocable Policy-Based Chameleon Hash. In: ESORICS. Springer International Publishing (2021)