RESEARCH-ARTICLE

Temporally-limited blind-regroup of anonymous credentials

LIQUN CHEN, University of Surrey, Guildford, Surrey, U.K.

CHIN HEI HO

MARK RYAN, University of Birmingham, Birmingham, West Midlands, U.K.

CHRISTOPHER WILLIAMSON

Open Access Support provided by:

University of Birmingham

University of Surrey

Published: 13 October 2025

Citation in BibTeX format

CCS '25: ACM SIGSAC Conference on Computer and Communications Security
October 13 - 17, 2025
Taipei, Taiwan

Conference Sponsors:
SIGSAC

https://doi.org/10.1145/3733802.3764056
ISBN: 9798400718984

Temporally-limited blind-regroup of anonymous credentials

Liqun Chen
University of Surrey
Guildford, United Kingdom
liqun.chen@surrey.ac.uk

Chin Hei Ho
SW7 Group
London, United Kingdom
eric.chinheiho@sw7group.com

Mark Ryan
University of Birmingham
Birmingham, United Kingdom
m.d.ryan@bham.ac.uk

Christopher Williamson
SW7 Group
Hong Kong, Hong Kong
christopher.williamson@sw7group.com

Abstract

We propose a new mechanism for digital identity schemes called 'blind-regroup', which, given a credential, allows a querying authority to identify all the existing credentials created by the same user. Unlike traceability, blind-regroup does not identify the ground identity of the user. Blind-regroup allows the user to continue creating credentials, and past blind-regroup queries do not compromise the anonymity of credentials created in the future. Blind-regroup is developed in a setting that satisfies authority transparency; that means that authorities can ultimately be held accountable for abuses of their power. We prove the correctness of blind-regroup (that is, it returns all and only all of the existing credentials matching the given credential); and we prove its security (namely, credentials not captured by blind-regroup remain unlinkable, and credentials are unforgeable). Our approach requires highly specialised zero-knowedge proofs; to demonstrate feasibility we provide SNARK implementations for these proofs and a performance analysis.

CCS Concepts

• **Security and privacy → Security services**; **Pseudonymity, anonymity and untraceability**; **Privacy-preserving protocols**;

Keywords

Security, privacy, unlinkability, accountability, digital identification, authentication

ACM Reference Format:
Liqun Chen, Chin Hei Ho, Mark Ryan, and Christopher Williamson. 2025. Temporally-limited blind-regroup of anonymous credentials. In *Proceedings of the 2025 Workshop on Privacy in the Electronic Society (WPES '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3733802.3764056

1 Introduction

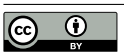
Anonymous credentials. Anonymous credential schemes [5] are concerned with striking an appropriate balance between privacy and accountability. For example, in the self-sovereign identity paradigm (SSI) [21], users are accountable in the sense that they may be required to prove an identity attribute prior to accessing an online service. Such users also enjoy privacy because different presentations of the same credential are unlinkable. A limitation of SSI is that it does not provide any ability for a relying party (such as an online service) to carry out investigations of a user, in the case of some misbehaviour.

The concept of traceability is a partial solution to this problem (e.g., [4, 14]). There are various definitions of traceability; in general schemes that offer this feature include an authority figure (which may be decentralised) that is empowered to make a query about the owner of a digital identity or credential. The output of this query will typically identify the *ground identity* of the credential owner, which had previously been registered by some credential issuer.

Traceability in the form described above is useful because it is a form of accountability that does not require the cooperation of the user. However, it has several limitations. First, it completely de-anonymises the user, which may be more than is needed for the investigation at hand. Second, it does not allow the tracing authority to find all the other credentials created by the user that is being investigated. Instead, the authority only learns the ground identity of the credential being queried. User activity identification - in which all activity of a user is identified as such - is another concept that partially addresses this weakness of traceability. However, it has a serious privacy limitation: once a user has been subject to activity identification, they are fully identified and may be unable to present their credentials again in an unlinkable way. Revocation - that is, preventing a user from acting - may also be too strong a sanction for some purposes.

Blind-regroup. We introduce a new query that addresses the limitations of traceability and revocation, called *blind-regroup*. We assume a digital identity scheme in which a user is capable of generating any number of credentials that are, by default, unlinkable to each other and to the user's ground identity. A blind-regroup query takes as input a credential, and produces as output the set of all credentials made by the same (unknown) user so far. The query does not reveal the ground identity of the user. Our paper focusses on blind-regroup, but we intend our technique to be integrated with other identity schemes that offer other query functionalities, such as identity schemes using randomisable signatures (e.g., [3], [18], [12]).

Temporal limitation. Blind-regroup returns the set of credentials that have been created by the user so far. After a blind-regroup query, the user can go on to create further credentials. An authority



WPES '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1898-4/25/10
https://doi.org/10.1145/3733802.3764056

that has done a blind-regroup query before those further credentials were created is not able to link those further credentials to each other or to the previous credentials, except by making a further blind-regroup query. Hence, blind-regroup allows authorities to make meaningful investigations without revealing ground identities and allows users to recover from an investigation and continue to enjoy privacy.

Authority transparency. Queries made by an authority should be accountable to users. Authority transparency means that users are able to access information about the queries that have been made (the granularity of that information is a parameter that can be set). We use the ideas of [19] to achieve authority transparency. More specifically, the technical capability to make certain decryptions is decentralised to a set of trustees which are instructed to respond only when there is a valid request from an authority that is allowed to make a query. This decentralised model works well with public ledger systems: each trustee may be instructed to reply only to requests made on the ledger, so that abuse of authority is necessarily overt.

Certonyms. In this paper, we use the term *certonym* ('certified pseudonym'). A certonym is a digital identity under the user's control, which (when there is probable cause or a legitimate legal basis) allows certain queries that can trace it to users or link it to other certonyms of the same user. The purpose of these queries is to allow enforcement of regulations. Crucially, the queries are only possible in certain circumstances, and only in a way that satisfies authority transparency by unavoidably leaving evidence of the linking. Certonyms enhance pricacy by scope-limiting and making observable the queries made about users.

This paper focusses on blind-regroup, which is an essential part of certonymity. The blind-regroup functionality we develop is 'backward-compatible' with existing forms of identity in the sense that certonyms can be cryptographically bound to another identity in order to augment that identity with blind-regroup capabilities.

1.1 Contributions

(1) We present a protocol that enables users to create multiple credentials (which we call *certonyms*), and enables an authority to perform temporally-limited blind-regroup queries, with authority transparency. Certonyms can enhance privacy in situations where authorities insist on some form of regulatability or observability.
(2) We prove the correctness of blind-regroup (that is, it returns all and only all of the existing credentials matching the given credential); and we prove its security (namely, credentials not captured by blind-regroup remain unlinkable, and credentials are unforgeable).
(3) We provide an initial implementation of all zero-knowledge proofs required by the protocol and discuss the scheme's efficiency and scalability. We use the Groth16 SNARK and choose elliptic curves that ensure compatibility with popular blockchains.

2 Preliminaries

2.1 Notation

Let $\mathbb{G}$ be a cyclic group of prime order p. We write the group operation multiplicatively and our group will later be instantiated using an elliptic curve. Let g be a generator of $\mathbb{G}$. For any $a \in \mathbb{Z}_p$ and $h \in \mathbb{G}$, we write h^a to denote the group operation applied to h repeatedly a times.

Let $\# : \{0,1\}^* \to \mathbb{Z}_p$ be a function that maps arbitrary-length bit strings to elements of $\mathbb{Z}_p$. The specific instantiation of # will be given later; as the notation suggests we need this function to be collision-resistant and will instantiate it with a properly chosen hash function.

Let $\nu : \mathbb{G} \to \mathbb{G}$ be a function defined by $\nu(h) = g^{\#(h)}$, where the function # is applied to h by parsing the group elements into bits. For a given input h and non-negative integer i, we use $\nu^i(h)$ to denote the recursive application of ν to h for i times; in particular $\nu^0(h) = h$ and $\nu^2(h) = \nu(\nu(h))$.

We will use $\mathbb{Z}_{\geq 0}$ to indicate the set of non-negative integers.

2.2 Parties to the protocol

- Issuer, an agent that acts as a verifier of ground (or legal) identities, and provides the means for users to obtain certonyms. Issuers deduplicate users to ensure that each holder of a legal identity can onboard only once.
- Users, individuals who obtain from Issuer the ability to create certonyms. Users may use certonyms to open an account or to certify that an existing blockchain address under their control.
- Relying Party (typically a service provider), an online platform that accepts a certonym from Issuer as a valid form of identity and is authorised to request a query in relation to a certonym.
- Trustees, parties that jointly participate in a transparent threshold decryption scheme, which enables an answer to a query about a certonym. Trustees are not required to exercise any judgment about the merits of decryption, and can be prevented from knowing any information about it.
- Ledger, a public append-only store of information which is not required to be trusted. The ledger stores two things: a list of all established certonyms; and a Merkle tree containing information about the linkability requests that have been made about certonyms.

Each deployment of the system specifies a single ledger and relying party (multiple relying parties can be supported using standard techniques beyond the scope of this paper). The system supports multiple issuers and multiple users but for simplicity we assume here a single issuer.

2.3 Dynamic threshold decryption

A threshold decryption scheme specifies a public key TPK and involves n trustees that jointly hold shares $\{sk_i\}_{i=1,\ldots,n}$ of a secret key. Trustees participate in decryption by publishing partial decryptions that depend on their secret key share and on the ciphertext to be decrypted. Decryption is successful as long as a threshold $t \leq n$ of them follow the protocol. Trustees decrypt in response to a request

of an external party that cannot directly participate in decryption. In our case, this external party is the relying party.

Dynamic threshold schemes were introduced in [10]. In such schemes, trustees need not participate in any multi-party ceremony to generate the threshold public key (avoiding many security pitfalls [9]). Moreover, encryptors can choose the set of decryptors and the decryption threshold. In certonymic practice, subject to potential constraints imposed by the relying party, this means users can select threshold decryption parameters (n, t). We construct a dynamic threshold decryption scheme using strong zero-knowledge machinery:

Local generation of threshold public keys. Encryptors generate their own threshold public keys and trustee key shares $\{sk_i\}_{i=1,\ldots,n}$ for use within a limited scope. In particular, users of the certonymic protocol generate a new TPK for each certonym and use it to produce each ciphertext contained within the certonym. Each share sk_i is encrypted individually to a trustee with public key PK_i. These encrypted shares, along with a zero-knowledge proof of correct construction, are then appended to the certonym as auxiliary information. Appendix B contains details of the method, and a summary is provided in Figure 1. We note that ciphertexts in our dynamic threshold scheme have size linear in n due to the individually encrypted sk_i values. Moreover, we use general-purpose SNARKs to prove correct computation of the sk_i. Besides the initial work [10] on dynamic threshold decryption schemes, works [6–8] improve ciphertext lengths at the cost of requiring encryptions be made over a pairing-friendly group, significantly restricting cryptographic flexibility. Most similar to our approach is that of [15], which uses NIZKs rather than general purpose SNARKs, improving performance, especially for protocols in which this dynamic threshold proofs are the bottleneck. We leave optimisations of this aspect of our mechanism to future work.

Encrypted trustee responses. Trustees publish encryptions (with respect to a public key of the relying party) of their partial decryptions so that only the relying party may derive the plaintext; this is discussed further in Section 2.4.

2.4 Transparent decryption

Transparent decryption is a type of protocol that cryptographically ensures the act of decryption is made necessarily overt. As in a threshold decryption scheme, this involves decentralisation in which the capability to decrypt is held jointly by a set of *trustees*. A transparent scheme requires that trustees will only act in response to the publication of a corresponding decryption request on an append-only ledger. This process ensures that decryptions are transparent to the people who are able to inspect the ledger. A decryption cannot have taken place unless the corresponding request has been published or a sufficient threshold of trustees do not follow the protocol. The protocol can be altered so as to vary the level of transparency of decryption requests, for example by encrypting queries to the trustees, implementing some form of time-delay, or re-randomising ciphertexts prior to their appearance in a decryption request. For simplicity, we describe a protocol without these measures in which requests are fully public.

Trustee key management is important; too many lost keys make queries impossible and too many leaked keys make accountability of queries impossible. We assume for now that no threshold-sized coalition of trustees collude or have had their keys compromised. In practice, key rotation techniques or proactive secret sharing may be used to reduce trustee-associated risk. Deployments may require that certonyms be refreshed from time-to-time, at a cadence proportional to trustee key churn.

Our system is designed in such a way that a blind-regroup query with respect to a certonym reduces to a request for decryption of a single ciphertext within that certonym. Trustees monitor a *ledger* for decryption requests, and publish responses to satisfy those requests. Trustees are trusted to publish if and only if they discover a new and valid request. The format of a decryption request is specified in Figure 5. Trustee responses are encrypted with a further key to limit who can derive the plaintext. Typically this will be a public key of the relying party, which is the authority that makes queries.

2.5 The ledger

The ledger serves two distinct purposes in our protocol, and hence stores two types of things. Firstly, it stores a list of all established certonyms. Secondly, the ledger maintains a Merkle tree that contains three types of leaves, with labels est, br and fup. A *certonym-leaf* is of the form (V, est), where V is a hash value. A certonym leaf is associated with a certonym and is added to the tree by users who are creating a new certonym. A *request-leaf* has the form (V, br) or (V, fup), and is associated with a blind regroup query request (initial or follow-up, respectively). Request leaves are added by either the Issuer or the Relying Party. The protocols for establishing and blind-regrouping certonyms rely on these items in the Merkle tree.

A Merkle tree is a binary tree in which data is stored at the leaf nodes, and the hash $H(d_\ell, d_r)$ is stored at each non-leaf node having children that store data d_ℓ and d_r. We assume that participants can access the ledger, which offers the following functions:

- root(), which returns the hash stored at the root of the ledger.
- add(x), which creates a new leaf node and adds it to the right of all the existing leaf nodes, and stores the data x there. The hashes stored at non-leaf nodes are updated as needed. The add function reorganises parts of the tree in such a way that the height of the tree storing n data items is bounded by $\log_2(n) + 1$.
- proof_of_presence(x, R), which returns data items stored in the ledger that are sufficient to prove that x was was present in the ledger when it had the root R.
- proof_of_extension(R, S), which returns data items stored in the ledger that are sufficient to prove that version of the ledger with root S is an add-only extension of its previous version which had root R.

These functions are standard; for example, they are functions used in the ledger logs of certificate transparency [1, 16]. We provide further information in Appendix E[1].

[1]This appendix is only in the long version of the paper, available on the authors' websites.

Overview of threshold decryption scheme:

- $\textsc{ThreshEnc}_{\boldsymbol{PK}}(n, t, l, M)$, done by user Alice:
  - *Input:* Public keys of trustees, presented as a vector $\boldsymbol{PK} = (PK_1, ..., PK_n)$, the threshold value t, and the plaintext vector $M = (M_1, ..., M_l)$, where $n \geq 1$, $t \leq n$ and $l \geq 1$.
  - *Method:*
    - Generate t-out-of-n threshold decryption keys $sk_1, \ldots, sk_n$ and their public key TPK (see Appendix B.1).
    - Compute the encrypted threshold key vector $\boldsymbol{E}$: $e_1 = \mathsf{Enc}_{PK_1}(sk_1), \ldots, e_n = \mathsf{Enc}_{PK_n}(sk_n)$, and π_{enc} a proof to show that $\boldsymbol{E}$ and TPK are correctly constructed. The proof makes use of randomnesses associated with the encryptions within $\boldsymbol{E}$.
    - Compute the ciphertext vector $C = (C_1, \ldots, C_l)$, where $C_i = \mathsf{Enc}_{TPK}(M_i)$, using fresh randomnesses for each encryption.
  - *Output:* $(\boldsymbol{E},\ TPK,\ \pi_{enc},\ C)$.
- $\textsc{TrusteeDec}_{SK_i}(e_i, C, RPK)$, requested by the relying party and done by Trustee i holding key (SK_i, PK_i):
  - *Input:* Trustee i's secret key SK_i, encrypted threshold key e_i which is the ith element of the vector $\boldsymbol{E}$, ciphertext C, and the relying party's public key RPK.
  - *Method:*
    - Compute $sk_i = \mathsf{Dec}_{SK_i}(e_i)$.
    - Compute a share of the plaintext vector $m_i = \mathsf{PartDec}_{sk_i}(C)$ (see Appendix B.2 for definition of PartDec).
    - Compute the encrypted share $c_i = \mathsf{Enc}_{RPK}(m_i)$.
  - *Output:* c_i.
- $\textsc{Combine}((c_i)_{i \in I})$, done by the relying party holding key (RSK, RPK):
  - *Input:* a threshold number $|I| \geq t$ of the encrypted shares $(c_i)_{i \in I}$.
  - *Method:*
    - Decrypt each share $m_i = \mathsf{Dec}_{RSK}(c_i)$ $(i \in I)$.
    - Combine the shares $(m_i)_{i \in I}$ to get the decrypted message by computing $M = \mathsf{Interpolate}((m_i)_{i \in I})$ (see Appendix B.3 for definition of Interpolate).
  - *Output:* M.

Figure 1: Threshold cryptosystem summary. Trustee i has public key PK_i and private key SK_i. TPK is an ephemeral threshold public key created by Alice to encrypt the messages $M = (m_1, \ldots, m_l)$; the secret keys sk_i are sent to the trustees encrypted by their public keys. The relying party with public key RPK can request a decryption; it will receive partial decryptions (encrypted with RPK) from a threshold number of trustees, and, after decrypting them with its secret key RSK, it can combine them.

3 Certonyms and their protocols

3.1 Certonyms and blind-regroup queries

We propose a model of digital identity which provides users a high degree of privacy and autonomy by default, while also ensuring some level of regulatability and accountability. We use the term 'certonym' to represent a digital identity within this model. In the certonymic model, a designated relying party may make certain queries about the underlying holder of a certonym or about the relationships between certonyms. Core to certonymity is the idea that authorities can be held accountable for queries that they make about certonyms and users. This helps to avoid authority overreach.

In this paper, we focus on how to implement a *blind-regroup query*. The input to this query is a certonym. The output is the set of all certonyms currently controlled by the same person that controls the input certonym. The query is temporally-limited in the sense that the user can continue to make new certonyms; by virtue of being created after the query execution finished, these certonyms cannot be linked to the set of certonyms that was output by the query unless a subsequent blind-regroup query is performed.

We emphasise that a certonymic protocol with blind-regroup will likely be augmented with other queries and the set of queries will enable proportionate investigations into user behaviour that reveal approximately the minimum information required to satisfy the goals of a typical investigation.

We expect each user to generate many certonyms and for there to be a large universe $\mathcal{U}$ of existent certonyms. As mentioned earlier, a more basic certonymic query is to test whether two certonyms are controlled by the same user. We note that this functionality is already sufficient to implement the blind-regroup functionality: the relying party can perform $|\mathcal{U}| - 1$ such queries to test whether the input certonym has the same controller as another certonym, for each other certonym in $\mathcal{U}$. Executing $\approx |\mathcal{U}|$ queries is impractical except for very small deployments of the certonymic protocol. The core of this paper is to describe a realisation of blind-regroup that requires only one query, along with a reasonable amount of post-processing by the relying party.

3.2 The protocols

Our certonym scheme includes several protocols, which will be introduced in this section. All those protocols will make use of global parameters as shown in Table 1.

Protocol Issue*: onboard with Issuer.* This is a one-time protocol in which Alice approaches the Issuer with identity documents. The Issuer verifies that Alice has not previously onboarded and makes

g	Cryptographic group generator for $\mathbb{G}$
$\boldsymbol{PK}$	Public keys of trustees
MAX_q	Maximum number of *blind-regroup* (BR) queries w.r.t. a given ground identity
MAX_c	Maximum number of certonyms that a user may create in a given generation (meaning between blind-regroup queries pertaining to their ground identity)

Table 1: Table of global protocol parameters

(1) Alice approaches Issuer with her ground identity "AliceID". Issuer verifies Alice's identity documents to confirm that Alice has not performed Issue previously.
(2) Issuer and Alice jointly contribute randomness in such a way that only Alice learns the outcome:
  (a) Alice privately selects a random nonce $nonce_{Alice}$.
  (b) Alice computes a commitment Com to $nonce_{Alice}$ and sends this to the Issuer.
  (c) Issuer independently selects a random nonce $nonce_{Issuer}$ and sends this to Alice in the clear.
  (d) Alice computes $h_0 = v(nonce_{Alice}) \cdot v(nonce_{Issuer})$.
(3) Alice computes the values $h_i = v(h_{i-1})$ (for $1 \leq i \leq MAX_q + 2$).
(4) Alice computes the value $V_{setup} = \#(h_{MAX_q+2}, 0)$.
(5) Alice computes a zero-knowledge proof π_{issue}, as defined in Figure 7.
(6) Alice provides V_{setup} and π_{issue} to Issuer, which verifies the proof.
(7) The Issuer accesses the ledger and performs a blockchain transaction to add (V_{setup}, est) as a certonym-leaf and (V_{setup}, br) as a request-leaf in the Merkle tree T on the ledger.

At the end of this protocol, Alice stores the following data locally:

$$(\{h_i\}_{0 \leq i \leq MAX_q+2},\ V_{setup})$$

Figure 2: The Issue protocol.

a record of her involvement with the protocol. In practice, she will use a passport or government issued ID to establish her ground identity. Alice and Issuer engage in a two party computation, at the end of which Alice gains the capability to create certonyms. The full Issue protocol is defined in Figure 2.

Protocol Establish*: create a certonym.* Alice computes a new certonym by using the secret values generated from the Issue protocol and the public information stored on the ledger.

- The certonym can be overtly cryptographically linked to the Issuer, to enable trust in the integrity of the initial onboarding process.
- No coalition of parties, absent a sufficient threshold of trustees, can link Alice's certonym with Alice's ground identity.
- No coalition of parties, absent a sufficient threshold of trustees, can link any two of Alice's certonyms with each other on the basis of their owner's common ground identity.

This protocol is defined in Figure 3. Users monitor for blind-regroup queries related to any certonym they control and maintain an integer β to represent how many such queries have occurred. This value will be used in the protocol.

Protocol Blind-regroup*: Given a certonym, find all other certonyms associated with the same ground identity.* This protocol is run between a relying party and a set of trustees.

(1) This protocol must not reveal any information about the ground identity.
(2) The total trustee effort in the protocol must be independent of the total number of certonyms that have been generated.
(3) Suppose that this protocol has been run, and an equivalence class of certonyms has been identified, all of which are associated with the same (unknown) ground identity. Suppose further that the user with that ground identity creates a new certonym. Then that certonym should not be linkable to the equivalence class until this protocol has been run again.

This protocol is defined in Figure 4. We think of each certonym of a given user as belonging to a certain generation, where generations are bookended by blind-regroup queries pertaining to the user. The generation of a certonym is not detectable by default. A user's first certonyms will belong to generation 0. At any later time, a freshly generated certonym will belong to generation β, where β is the number of previous blind-regroup queries that have ever pertained to the user at that time. At most MAX_c certonyms may be created by a user in a given generation.

3.3 The construction in a nutshell

The core idea of the construction is that each user creates certonyms within a given (user-specific) *generation* and then proceeds to the next generation exactly when any of their certonyms has been the subject of a blind-regroup query. An effect of the query is that all certonyms of the user from the outgoing (and all prior) generations become linkable from the perspective of the relying party. An effect of the user moving to the next generation is that newly-created certonyms cannot be linked to those of earlier generations unless there is another relevant blind-regroup query.

Users are prevented from skipping generations, because this would break the blind-regroup functionality. As such, users are unable to create certonyms from within a given generation unless they prove that they are not skipping a generation and that the certonyms in their current generation have been linked by a BR query . Users produce zero-knowledge proofs of these conditions, which include proofs of presence of items in the ledger Merkle tree. The Merkle tree proofs establish that certain certonyms have been created in the past and that certain BR queries have been executed in the past. The zero-knowledge proof protects the user from revealing which such past certonyms and queries are relevant to their new certonym, and also checks that the rules about certonymic generations are obeyed. We stress that the relying party is not required to be trusted to provide Alice her ability to create unlinkable credetials after a BR query: each query must be recorded on the

ledger, leaving Alice with the information she needs to justify the creation of her next generation of certonyms.

Within a given generation, users can create multiple certonyms that are all unlinkable prior to the next blind-regroup query. Ciphertext re-randomisation and a hash function mechanism are the tools used to ensure that certonyms of the same generation for the same user remain unlinkable by default.

3.3.1 An illustrative construction. In this section we describe a simplified certonym construction. A downside of this simplified construction is that the size of the certonym (and the number of decryptions per query) grows linearly in MAX_q, the total number of blind-regroup queries that can be made with respect to a given user. Some of the complexities of the full protocol are motivated by the goal of reducing the size of the certonym to be independent of MAX_q.

For our illustrative purposes, a simplified certonym has the following components:

$$(H_0, ..., H_{MAX_q+1}, V, W, y, \pi)$$

First, we explain the values $H_0, ..., H_{MAX_q+1}$. The certonym protocol begins with the generation of a pseudorandom sequence of values $h_0, ..., h_{MAX_q+2}$. No one party can control the values in this sequence: it is the result of jointly-chosen randomness sourced from a user Alice and the Issuer. The sequence values are elements of a cryptographic group and they are linked by the ν function, which is a hash-to-group function that is easy to compute but hard to invert. We define this function in Section 2.1 and give further details in Section 6.1. For all $0 \leq i \leq MAX_q + 1$, we have that H_i is the encryption of h_i.

The values $\{h_i\}_{i=1,...,MAX_q+1}$ are in a one-to-one correspondence with certonym generations. Initially, Alice has been the subject of no blind-regroup query and her certonyms are of the generation associated with the value h_{MAX_q+1}. The Merkle tree on the ledger contains leaves, placed there by the Issuer during the Issue protocol, that allow Alice to prove that she may create certonyms in this first generation. In effect, those leaves are dummy leaves that use the value h_{MAX_q+2} to "fool" the system into believing that a prior BR query justifies Alice's creation of certonyms in this generation, even though in the case of the first generation no such BR query has occurred. The upshot of this is that first-generation certonyms are indistinguishable from non-first-generation certonyms. Meanwhile, the absence of any other leaves regarding BR queries relevant to Alice ensure that she may not yet move beyond the first generation. As she creates multiple certonyms from this initial generation, Alice re-randomises the ciphertexts $\{H_i\}_{i=0,...,MAX_q+1}$ to break linkability between certonyms and proves within π that the certonym is constructed correctly. The value V is the hash of a concatentation of h_{MAX_q+1} and some other value of sufficiently low entropy: an integer at most MAX_c. V is used by the relying party to link certonyms of the same generation at the time of a blind-regroup query. In particular, the relying party decrypts h_{MAX_q+1} and brute-forces the low-entropy value. As such, setting MAX_c to be large gives Alice more room to create certonyms in a single generation but also increases the BR query post-processing effort of the relying party. The values W and y serve to bind a signing and verification key pair to the certonym.

Now, if the relying party requests a first blind-regroup query with respect to this certonym, a new leaf recording this event is added to the Merkle tree. This allows Alice to move to the next generation: Alice will use the existence of this new leaf to prove that a first BR query has occurred with respect to her and that she is entitled to proceed to the second generation, in which h_{MAX_q} will be used in the computation of V. Meanwhile, as a result of the query, the relying party learns h_{MAX_q+1} and uses this to link all certonyms Alice created prior to the query.

Later, if the relying party wants to follow-up on Alice and issues another blind-regroup query, the same process repeats: the relying party will use h_{MAX_q} to link Alice's certonyms and Alice will then use h_{MAX_q-1} to make new ones. In general, if the blind-regroup protocol has been executed β times, then Alice uses $h_{MAX_q+1-\beta}$ in newly created certonyms' V value. After MAX_q BR queries, certonyms use h_1 to create the V value. Even though h_0 is never used to form the V value of a certonym of any generation, it is needed in the full version of the scheme, in which certonyms only contain two encrypted values from the set $\{H_i\}_{i=0,...,MAX_q+1}$, so that certonyms in the final generation have the same format as all other certonyms.

3.4 Efficiency of blind-regroup

Blind-regroup is designed so that all certonyms of a given user, up to the present moment, can be identified via the act of a single threshold decryption done by trustees. In exchange for this property, there is a post-processing step done by the relying party.

The most intensive aspect of this post-processing step is the computation of $\mathsf{Cfrom}(h)$ for some decrypted value h. This involves computing $\mathsf{Vfrom}(h)$ and checking each certonym to see whether its V value is an element of this set.

This computation of $\mathsf{Vfrom}(h)$ requires MAX_q-wise recursive computation of the function ν as applied to h. For each such recursive step, MAX_c hashes are computed; the total hashing burden grows like $O(MAX_q \cdot MAX_c)$. Even for generous settings of these parameters, the hash burden is not high for a relying party, which we presume will have good computational resources.

To check which certonyms have a value in $\mathsf{Vfrom}(h)$, the relying party may employ techniques like a Bloom filter to keep computational costs manageable.

3.5 Using certonyms

Signing with certonyms. A user may use their certonyms to sign data. A relying party is a platform that accepts certonyms as an ideneity form and accepts signatures by certonyms, perhaps depending on the associated Issuer. Users can establish that they control a certonym by signing a message with it. The message can be generated by the user, or it could be a challenge sent to the user by a relying party.

Suppose Alice has a certonym $(G,\ H,\ V,\ W,\ y,\ \pi)$.

(1) Alice signs the message using her signing key x corresponding to the certonym's verification key y, producing a signature (e.g., a Schnorr signature).
(2) Alice sends the signature together with her certonym to the relying party.

(1) Alice observes the ledger and counts the number of blind-regroup (BR) queries made with respect to her certonyms. Let this nonnegative integer be β. If $\beta > MAX_q$, then Alice aborts.
(2) Alice creates a certonym-specific Schnorr signing and verification key pair (x, y) where $y = g^x$.
(3) Alice computes $\textsc{ThreshEnc}_{\boldsymbol{PK}}(n, t, 2, M = (h_{MAX_q-\beta}, h_{MAX_q-\beta+1}))$, resulting in
$$(\boldsymbol{E}, TPK, \pi_{enc}, C = (H_{MAX_q-\beta}, H_{MAX_q-\beta+1})),$$
where the randomnesses of encryption are $r_{MAX_q-\beta}$, $r_{MAX_q-\beta+1}$, respectively.
(4) Alice arbitrarily chooses a positive integer $\epsilon \leq MAX_c$. Alice computes two values $V = \#(h_{MAX_q-\beta+1}, \epsilon)$ and $W = \#(h_{MAX_q-\beta+1}, y)$.
(5) Alice accesses the ledger and observes the state of its Merkle tree. Either:
- Alice finds a request-leaf value $(\#(\nu(h_{MAX_q-\beta+1}), \epsilon'), \text{br})$ for some ϵ', or
- Alice finds a request-leaf value $(\#(\nu(\nu(h_{MAX_q-\beta+1})), \epsilon'), \text{fup})$, and a certonym-leaf value $(\#(\nu(h_{MAX_q-\beta+1}), \epsilon''), \text{est})$ for some ϵ' and ϵ''.

If Alice cannot find the required leaves, she aborts.
(6) Alice then sets T_R to be the root `root()` of the ledger's Merkle tree and constructs a proof of presence of the leaves in the Merkle tree with that root.
(7) Alice computes a zero-knowledge proof π_{est}, as defined in Figure 9.
(8) The certonym is
$$C = (H_{MAX_q-\beta},\ H_{MAX_q-\beta+1},\ V,\ W,\ y,\ \pi_{est}).$$
Note that only Alice knows x, which is the signing key associated with the certonym.
(9) Alice submits the certonym to the ledger, which then verifies the certonym. If the verification passes, the ledger stores the entire certonym C on the ledger and adds the certonym-leaf (V, est) to the ledger Merkle tree.

Figure 3: The Establish protocol

Certonymization of an existing identity. In some circumstances, Alice will want to associate her certonym with an established account on a platform. The account will typically already be associated with a form of identity that is native to the platform. In this scenario, the relying party challenge may contain a reference to the account: this means that the certonym signature will establish that the owner of the certonym wishes to be associated with the referenced account. To ensure that the account owner is the same person as the controller of the certonym, the platform may require that the user log-in to her account and confirm its association with the given certonym.

Certonym signature verification. Certonyms can be verified by the relying party. We note here, and will emphasise in the implementation section, that certonyms can be verified on a smart contract on sufficiently expressive blockchains, such as those using the Ethereum Virtual Machine.

When a relying party is presented with a signature σ and a certonym $C = (G,\ H,\ V,\ W,\ y,\ \pi)$:

- It verifies the signature σ using the public key y;
- It tests the validity of the certonym by confirming that the proof π is valid, and
- the tree root T_R used as a part of the public witness of π is extended by the current tree root $root()$ of the ledger.

3.6 Extensions and variations

Identity attributes. Encryptions of identity attributes can be included within (and cryptographically bound to) a certonym. For instance, the user's citizenship status may be checked by the Issuer and encrypted to another ciphertext in the certonym. These can be decrypted upon request by relying party or users can issue proofs of the decrypted value, as in SSI protocols.

Full user decryption, or testing equality of underlying identity. The scheme can readily be extended to enable a query that extracts the user identity from a certonym, or the query that determines whether two certonyms are owned by the same user. This can be achieved by appending to the certonym the randomised encryption of an identifier unique to the user, with appropriate cryptographic binding.

Revocation of certonym creation privilege. As a variant, the relying party may want to temporarily block a user Alice from creating more certonyms after a BR query. This requirement could be represented in the BR query request leaf. As a result, proofs π_{est} would prove that the Merkle tree leaves relevant to a newly proposed certonym are not marked with this requirement. Such an act of revocation of privileges by the relying party is detectable by anyone with read access to the ledger and can be publicly reversed by the relying party at any time.

4 Correctness proofs

Suppose user A has created certonyms $C_1, \ldots, C_n$, and then a blind-regroup query is done on C_i (for some $1 \leq i \leq n$). Suppose the user goes on to create certonyms $C_{n+1}, \ldots, C_m$. The data that was returned by the blind-regroup query allows one to link all the certonyms in $C_1, \ldots, C_n$ with each other, but none of the certonyms in $C_{n+1}, \ldots, C_m$ with any other certonym. We provide proofs of 'completeness' (meaning the blind-regroup query produces no false negatives) and 'soundness' (meaning no false positives of the blind-regroup query). Additionally, we prove 'executability' (meaning that a user can execute the establish protocol).

4.1 Proof of executability

The Establish protocol lays out the various steps that a user Alice must take to generate a new certonym. It is immediate that all of these steps are feasible for Alice to complete, other than Step 5, in which she must find Merkle tree leaves with certain properties. Therefore we prove that Alice will always be able to find the leaves

Input: a certonym $C = (G,\ H,\ V,\ W,\ y,\ \pi)$.
Output: the set of certonyms created so far that have the same ground identity as C.
Notation: *All* is the set of all existing certonyms, which is derivable by viewing the ledger.
$\mathsf{Vfrom}(h) = \{\#(v^i(h), \epsilon) \mid 0 \le i \le MAX_q,\ 1 \le \epsilon \le MAX_c\}$
$\mathsf{Cfrom}(h) = \{(G, H, V, W, y, \pi) \in All \mid V \in \mathsf{Vfrom}(h)\}$

A set S is maintained by the relying party (RP). It is initially the empty set; $h \in S$ means that h has been uncovered by the RP during a blind-regroup (BR) query.

---

if $\exists h \in S$ such that $V \in \mathsf{Vfrom}(h)$ **then**
  $j \leftarrow \max_{j \in \mathbb{Z}_{\ge 0}} \{\exists h' \in S : v^j(h') = h\}$
  Pick h' arbitrarily from $\{h' \in S : v^j(h') = h\}$
  if $\exists (V', \text{est}) \in$ MT such that $\exists \epsilon : V' = \#(h', \epsilon)$ **then**
    Let C' be the certonym corresponding to (V', est)
    Parse C' as $(G',\ H',\ V',\ W',\ y',\ \pi')$
    Construct π_{fup}, as defined in Figure 8
    Submit request $(C', \mathsf{fup}, \pi_{\mathsf{fup}}, RPK)$ to the ledger
    View query responses $M_1, ..., M_n$ published by trustees
    Compute $h'' = \textsc{Combine}((M_1, ..., M_n))$
    Add h'' to S
    Return the result $\mathsf{Cfrom}(h'')$
  else
    Return the result $\mathsf{Cfrom}(h')$
  end if
else
  Submit request (C', br, RPK) to the ledger
  View query responses $M_1, ..., M_n$ published by trustees
  Compute $h' = \textsc{Combine}((M_1, ..., M_n))$
  Add h' to S
  Return the result $\mathsf{Cfrom}(h')$
end if

Figure 4: The blind-regroup (BR) protocol, which allows the relying party to obtain all the certonyms created by the user that owns a given certonym.

needed to generate a certonym. In Appendix F.3[2], we will prove the following claim:

Claim 1. *In Step 5 of Figure 3, Alice will never abort as long as she has read-access to the ledger.*

4.2 Proof of completeness

Theorem 1. *When a blind-regroup (BR) query is executed with respect to certonym $C = (G, H, V, W, y, \pi)$, the Issuer will be able to link together all certonyms created by the same owner as the owner of C.*

The proof is given in Appendix F.4[3].

[2]This appendix is only in the long version of the paper, available on the authors' websites.
[3]This appendix is only in the long version of the paper, available on the authors' websites.

In blind-regroup, the relying party publishes on the ledger a decryption request of the form (C, br, RPK) or $(C, \mathsf{fup}, \pi_{\mathsf{fup}}, RPK)$, where $C = (G, H, V, W, y, \pi)$ is a certonym, π_{fup} is a zero-knowledge proof, and RPK is a public key of the relying party. In its routine monitoring of the ledger, trustee i finds this request and responds as follows:

(1) Check that C is a valid certonym. If not, then halt.
(2) Parse the public inputs of proof π within certonym C to obtain $\boldsymbol{E}$, with ith element given by e_i.
(3) If the request is (C, br, RPK):
  (a) Check whether (V, br) is a Merkle tree leaf and add it if it is not.
  (b) Publish to ledger $M_i = \textsc{TrusteeDec}_{SK_i}(e_i, H, RPK)$.
(4) If the request is $(C, \mathsf{fup}, \pi_{\mathsf{fup}}, RPK)$:
  (a) Check whether (V, fup) is a Merkle tree leaf and add it if it is not.
  (b) Check the validity of π_{fup}.
  (c) Publish to ledger $M_i = \textsc{TrusteeDec}_{SK_i}(e_i, G, RPK)$.

Figure 5: The decryption API for blind-regroup

4.3 Proof of soundness

We prove that the result of an honestly issued query executed with respect to a certonym C contains only certonyms made by the same user as the user that made C. We work in the random oracle model. Let R be the cardinality of the range of the hash function. We prove our result in the simplest non-trivial case of two users; the bound can easily be extended to arbitrarily many users.

Theorem 2. *Suppose that a BR query targets a certonym owned by u_1. If $u_1 \neq u_2$ and users u_1 and u_2 do not collude, then, in the course of the query, no certonym owned by u_2 will be identified, except with probability at most $O\left(\frac{(MAX_c \cdot MAX_q)^2}{R}\right)$.*

The intuition behind the theorem is that if h_{0,u_1} and h_{0,u_2} are chosen randomly, then the sets $\{\#(v^i(h_{0,u_1}), j) \mid 0 \le i \le MAX_q,\ 1 \le j \le MAX_c\}$ and $\{\#(v^i(h_{0,u_2}), j) \mid 0 \le i \le MAX_q,\ 1 \le j \le MAX_c\}$ intersect with negligible probability. A formal proof is given in Appendix F.5[4].

5 Security proofs

We now provide security analysis of a certonym that has the form:

$$(G, H, V, W, y, \pi)$$

5.1 Oracles

Before discussing anonymity & unlinkability and unforgeability in the following subsections, we first introduce the oracles used in the security analysis of these two properties. For each oracle, the adversary $\mathcal{A}$ acts as a requester and the simulator $\mathcal{S}$ as a responder, so we say that $\mathcal{S}$ runs oracles for $\mathcal{A}$ to query. Aligned with the oracles, $\mathcal{S}$ maintains a user list U, which records the transcripts of running the oracles.

[4]This appendix is only in the long version of the paper, available on the authors' websites.

- O_{CrU} (create user): On input $u \notin U$, $\mathcal{A}$ requests that u is created. $\mathcal{S}$ creates the user u by using Issuer's function, stores u along with the user's keys in U and marks that u is honest. Note that $\mathcal{S}$ does not provide the user's secret key to the adversary. In the certonym scheme, a user creates and holds their key. The adversary is not assumed to access an honest user's key, so the O_{CrU} oracle follows this reality correctly.
- O_{CoU} (corrupt user): On input $u \in U$, $\mathcal{A}$ requests that the user u is corrupted. $\mathcal{S}$ discloses u's keys to $\mathcal{A}$ and marks that u is corrupted.
- O_{CC} (create certonym): On input $u \in U$, $\mathcal{A}$ requests that a certonym for u is created. $\mathcal{S}$ creates a certonym C, stores it in U and returns it to $\mathcal{A}$. For the same reason as the O_{CrU} oracle, $\mathcal{S}$ does not provide the secret key associated with the certonym to the adversary.
- O_S (sign message): On input certonym C and message m, $\mathcal{A}$ requests that the message is signed by using the certonym C and its corresponding key. $\mathcal{S}$ returns a signature s on the message m signed by using the certonym C and the corresponding key.
- O_{BR} (blind-regroup): On input certonym C, $\mathcal{A}$ requests that the blind-regroup protocol for this certonym is run. $\mathcal{S}$ returns the transcript of the blind-regroup protocol to $\mathcal{A}$.

5.2 Anonymity and Unlinkability

The property of certonym anonymity & unlinkability expresses the two facts:

(1) Anonymity: a user using their certonym is indistinguishable from any other user using their certonym.
(2) Unlinkability: If a user uses a certonym in one session, and another certonym in another session, and the certonyms have not been associated with a blind-regroup query, then the two sessions are unlinkable.

To express this property more formally, we consider an experiment, $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$, between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. $\mathcal{A}$ controls Issuer and $\mathcal{A}$'s actions are bounded in polynomial-time. $\mathcal{S}$ maintains a user list U, which includes each user's name, keys, certonyms and their states (honest or corrupted). U is empty when the experiment starts. $\mathcal{S}$ runs the following oracles for $\mathcal{A}$ to query: O_{CrU} (create user), O_{CoU} (corrupt user), O_{CC} (create certonym), O_S (sign message) and O_{BR} (blind-regroup).

The experiment includes three phases:

Phase 1: $\mathcal{A}$ can call all the oracles defined in Subsection 5.1 adaptively.

Challenge phase, which starts at the end of Phase 1:

(1) $\mathcal{A}$ chooses two honest users A and B and sends $\{A, B\}$ to $\mathcal{S}$.
(2) $\mathcal{S}$ selects a random element r from $\{A, B\}$.
(3) $\mathcal{S}$ generates a certonym C by calling $O_{CC}(r)$.
(4) $\mathcal{S}$ returns C to $\mathcal{A}$.

Phase 2, which starts at the end of the Challenge phase:

(1) $\mathcal{A}$ can continue to use the oracles with arbitrary arguments as in Phase 1, except that they can't call O_{CoU} with input A or B, or O_{BR} with input C.
(2) Particularly, $\mathcal{A}$ can call $O_{CC}(A)$ or $O_{CC}(B)$ multiple times.
(3) $\mathcal{A}$ can also call O_S for A or B, including $O_S(C, m)$ with a message m at $\mathcal{A}$'s choice.
(4) At the end of Phase 2, $\mathcal{A}$ outputs a value r'. It wins the experiment if $r' = r$.

We denote the fact that the adversary wins the above game by $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}(\lambda) = 1$.

Definition 1. *(Anonymity & Unlinkability). Given a security parameter λ, it is said that a certonym scheme holds the property of Anonymity & Unlinkability, if for any polynomial-time adversary $\mathcal{A}$, the probability of $\mathcal{A}$ winning the Anonymity & Unlinkability (AU) experiment is as follows:*

$$Succ^{AU}_{cert}(\mathcal{A}(\lambda)) = Pr[\mathsf{Exp}^{AU}_{cert,\mathcal{A}}(\lambda) = 1] \leq 1/2 + negl(\lambda).$$

Definition 2. *(Encryption with randomisable ciphertexts). It is said that an encryption scheme supports randomisable ciphertexts, if given a ciphertext C associated with a plaintext P and a key, one can create another ciphertext C', which will be decrypted to the same P by using the same key.*

Definition 3. *(Ciphertext indistinguishability of encryption with randomisable ciphertexts). This property is defined by using an experiment $\mathsf{Exp}^{CipherInd}_{Trustee}$, which is run between a polynomial-time adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. $\mathcal{A}$ has access to a target public encryption key (via creating a user query), so $\mathcal{A}$ can make ciphertexts for any messages under this key. In addition, $\mathcal{A}$ can make multiple queries about ciphertexts of $\mathcal{A}$'s choice to obtain the corresponding plaintexts. The queries are answered by $\mathcal{S}$. As a challenge, $\mathcal{A}$ provides two ciphertexts, C_0 and C_1, to $\mathcal{S}$. $\mathcal{S}$ randomly picks a bit $b \in \{0, 1\}$, randomises C_b to obtain a randomised version C'_b. $\mathcal{S}$ returns it to $\mathcal{A}$. $\mathcal{A}$ can then carry on making the queries as before except for the decryption query for C'_b. At the end, $\mathcal{A}$ outputs b'. If $b' = b$, $\mathcal{A}$ wins the experiment. It is said that an encryption scheme with randomisable ciphertexts holds ciphertext indistinguishability, if the probability of the aforementioned adversary succeeding is not more than a half plus a negligible value.*

Theorem 3. *For a suitable security parameter, λ, the certonym scheme is anonymous and unlinkable if:*

- *The underlying trustee encryption scheme used to create a certonym supports randomisable ciphertexts and ciphertext indistinguishability.*
- *The proof π is a Non-Interactive Zero-Knowledge Proof (NIZKP), which can be simulated. By simulated, we mean there is a simulation oracle to create a simulated π without knowing its secret input. The simulation oracle is accessible by the simulator $\mathcal{S}$. From a verifier's point of view, a simulated proof is indistinguishable from a real proof computed using its corresponding secret.*
- *The hash function used in the Schnorr signature is a random oracle.*

Proof. $\mathcal{S}$ performs the $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$ experiment with $\mathcal{A}$ as defined before, in which $\mathcal{S}$ responds to queries made by $\mathcal{A}$. The hash function used in the Schnorr signature scheme is through a random oracle model, which is run by $\mathcal{S}$. $\mathcal{S}$ has access to the simulation oracle of π to generate a verifiable zero-knowledge proof. To guarantee consistency between answers to various queries, $\mathcal{S}$ maintains

a certonym user list U, which includes each user's name, keys, certonyms and their states (honest or corrupted), and all queries and the corresponding answers associated with this user. The list U is empty when the experiment starts.

While $\mathcal{A}$ has the target of winning the experiment $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$, $\mathcal{S}$ has the target of breaking the ciphtertext indistinguishability of the underlying trustee encryption scheme. For this purpose, during the run of the $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$ experiment, $\mathcal{S}$ simultaneously runs $\mathsf{Exp}^{CipherInd}_{Trustee}$ experiment with their simulator. In this experiment, $\mathcal{S}$ plays the role of an adversary, who is allowed to ask the queries for creating a set of honest users (i.e., trustees), and decrypting a ciphertext. As a challenge, $\mathcal{S}$ submits two ciphertexts C_0 and C_1 to the simulator of $\mathsf{Exp}^{CipherInd}_{Trustee}$ and receives a randomised ciphertext C_b for $b \in \{0,1\}$. $\mathcal{S}$ needs to output b' and wins the experiment $\mathsf{Exp}^{CipherInd}_{Trustee}$ if $b' = b$.

In the $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$ experiment, $\mathcal{A}$ controls the Issuer but not the trustees. At the outset of the experiment, $\mathcal{S}$ runs Setup to create a set of trustees. To set up each trustee, $\mathcal{S}$ asks a query for creating an honest user to the simulator of the $\mathsf{Exp}^{CipherInd}_{Trustee}$ experiment, $\mathcal{S}$ receives a public key for this trustee and $\mathcal{S}$ shares the key with $\mathcal{A}$. $\mathcal{S}$ also runs Setup (or takes $\mathcal{A}$'s input) to create an Issuer, which has a public and secret key pair. All the values of the public and secret keys of the Issuer are known to $\mathcal{A}$.

In Phase 1, $\mathcal{S}$ handles the oracle queries listed in section 5.1 as follows.

- O_{CrU} (create user): To create an honest certonym user u, $\mathcal{A}$ sends $O_{CrU}(u)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $u \in U$. If yes, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ runs the *Issue* protocol with $\mathcal{A}$, in which $\mathcal{A}$ plays as an Issuer and $\mathcal{S}$ plays as u. $\mathcal{S}$ uses the trustees' public keys to generate related ciphertexts. At the end of the protocol, $\mathcal{S}$ obtains and stores $((h_i, H_i)_{0 \le i \le MAX_q+1}, \sigma_{MAX_q}, V_{setup})$. $\mathcal{S}$ records the transcript of this oracle run in U and marks that u is honest.
- O_{CoU} (corrupt user): To corrupt a certonym user u, $\mathcal{A}$ sends $O_{CoU}(u)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $u \in U$ and is marked as honest. If not, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ discloses u's secret key x together with the Issue protocol output $((h_i, H_i)_{0 \le i \le MAX_q+1}, \sigma_{MAX_q}, V_{setup})$ to $\mathcal{A}$ and marks that u is corrupted.
- O_{CC} (create certonym): To obtain a certonym for an honest user u, $\mathcal{A}$ sends $O_{CC}(u)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $u \in U$ and is marked as honest. If not, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ runs the *Establish* protocol with $\mathcal{A}$. At the end of the protocol, $\mathcal{S}$ creates a certonym $C = (G, H, V, W, y, \pi)$, stores the certonym together with its secret key x (where $y = g'^x$) in U and returns the certonym to $\mathcal{A}$.
- O_S (sign message): To request signing a message m using the certonym C and its corresponding key, $\mathcal{A}$ sends $O_S(C, m)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $C \in U$ and its corresponding user u is marked as honest. If not, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ creates a signature on m using C and its secret key x. $\mathcal{S}$ records this result to U and returns the signature s to $\mathcal{A}$.
- O_{BR} (blind-regroup): On input certonym C, $\mathcal{A}$ requests that the blind-regroup protocol for this certonym is run. $\mathcal{S}$ asks the simulator of the $\mathsf{Exp}^{CipherInd}_{Trustee}$ experiment for decryption queries on the relevant ciphertexts in C, and then $\mathcal{S}$ receives a set of plaintexts, which are associated with all the certonyms created so far that have the same ground identity as C. $\mathcal{S}$ returns the result of the blind-regroup protocol to $\mathcal{A}$.

$\mathcal{A}$ decides the time when Phase 1 is complete and the challenge phase starts. In the challenge phase, $\mathcal{A}$ outputs two certonym users' names, say A and B, which have not been corrupted. $\mathcal{S}$ generates two set of ciphertexts C_0 and C_1, where $C_0 = (G_A, H_A)$ and $C_1 = (G_B, H_B)$. $\mathcal{S}$ sends them to the simulator of the $\mathsf{Exp}^{CipherInd}_{Trustee}$ experiment, from which $\mathcal{S}$ receives a randomised ciphertext C_b, where $b \in \{0,1\}$. $\mathcal{S}$ then generates a certonym C_r based on C_b. Clearly, if $b = 0$, $r = A$ and otherwise $r = B$. Following the definition of a certonym,

$$C_r = (G_r, H_r, V_r, W_r, y_r, \pi_r).$$

V_r and W_r are two hash outputs and $\mathcal{S}$ can handle them based on the random oracle. To do so, $\mathcal{S}$ chooses two random values as V_r and W_r, which are represented as $\#(a, \epsilon)$ and $\#(b, y_r)$. $\mathcal{S}$ does not know a or b, but since π is simulatable, the consistency of C_r can be maintained, i.e., from $\mathcal{A}$'s point of view, C_r is indistinguishable from a real C_r. However, there are two conditions: (i) C_A and C_B should use the same ϵ (this is achievable); and (ii) neither (a, ϵ) nor (b, y_r) should be used as an input of querying the hash function $\#$ (if this happens, $\mathcal{S}$ will abort). The randomisation of the ciphertext also updates the public verification key y_r, which can be retrieved from C_b, but $\mathcal{S}$ does not know its corresponding secret key x_r. The last part of C_r is an NIZKP π_r. $\mathcal{S}$ generates it by calling the simulation oracle of this proof. This can be achieved since it is assumed that π_r is a simulated proof. Finally, $\mathcal{S}$ sends C_r to $\mathcal{A}$.

One example of π_r is a simulation-extractable KZG polynomial commitment scheme [17], which is an extention of the KZG polynomial commitment scheme [13]. A univariate case (proving a single polynomial) of simulation-extractable KZG commitments is presented in Material C of [17] and a general case with multivariate polynomials is in Section 4 of this paper.

In Phase 2, $\mathcal{S}$ and $\mathcal{A}$ carry on the query and response process as in Phase 1. Again, $\mathcal{A}$ is not allowed to make any Corrupt query to either A or B. In addition, $\mathcal{A}$ is not allowed to query $O_{BR}(C_r)$. However, $\mathcal{A}$ is allowed to query $O_{CC}(A)$ or $O_{CC}(B)$ multiple times, and as well as to query $O_S(C_r, m)$ with a message m at $\mathcal{A}$'s choice. We now see how $\mathcal{S}$ handles these queries:

- $O_{CC}(A)$ (or $O_{CC}(B)$): $\mathcal{S}$ responds to this query by following the protocol correctly and returns to $\mathcal{A}$ with a newly generated certonym.
- $O_S(C_r, m)$ (sign a message on the behalf of the user r): To request signing a message m using the certonym C_r, which is the output of the challenge phase, which is associated with the user $r \in \{A, B\}$. $\mathcal{S}$ creates a Schnorr signature s on m under the key y_r. Note that $\mathcal{S}$ does not have access to the corresponding secret signing key x_r. In this case, $\mathcal{S}$ uses the random oracle model to simulate a valid Schnorr signature s. $\mathcal{S}$ records this result to U and returns the signature s to $\mathcal{A}$.

At the end of Phase 2, $\mathcal{A}$ outputs r'. If $r' = A$, $\mathcal{S}$ outputs $b' = 0$; otherwise, if $r' = B$, $\mathcal{S}$ outputs $b' = 1$. We argue that if $\mathcal{A}$ wins

the experiment $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$, $\mathcal{S}$ will win the experiment $\mathsf{Exp}^{cipherInd}_{Trustee}$, because C_b is used to create C_r. In order to increase the success rate, the $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$ experiment should be run multiple times and as well as the $\mathsf{Exp}^{cipherInd}_{Trustee}$ experiment runs.

We can see that the first fact (anonymity) is held because if the probability of $\mathcal{A}$ outputting the correct r is higher than $1/2 + negl(\lambda)$, the probability of $\mathcal{S}$ outputting the correct b' is also $1/2 + negl(\lambda)$. However, this contradicts the assumption that the underlying trustee encryption scheme supports randomisable ciphertexts and ciphertext indistinguishability.

The second fact (unlinkability) is also held because the $\mathsf{Exp}^{AU}_{cert,\mathcal{A}}$ experiment allows $\mathcal{A}$ to query O_{CC} for any honest users, including the challenging users A and B. As for any corrupted users, $\mathcal{A}$ can create their certonyms by itself. Moreover, $\mathcal{A}$ is allowed to query O_S for any certonyms, including the challenging certonym C_r. If $\mathcal{A}$ can link C_r with any of these queries, $\mathcal{A}$ must be able to output a correct r every time, which will break the anonymity. As discussed before, this can only happen with a negligible probability.

Since both the two facts are held, the theorem follows. □

In Appendix C, we demonstrate how the modified KZG scheme is simulatable.

5.3 Unforgeability

This property guarantees that an adversary cannot create certonyms or signatures that would be linked to or attributed to a certonym of a different user who acts honestly.

To formalise this property, similar to the property of anonymity & unlinkability from section 5.2, we also consider an experiment, $\mathsf{Exp}^{\text{Unforge}}_{\text{cert},\mathcal{A}}$ between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. $\mathcal{S}$ maintains a user list U, which includes each user's name, keys, certonyms and their states (honest or corrupted). U is empty when the experiment starts. In this experiment, $\mathcal{A}$ controls the trustees, so the O_{BR} (blind-regroup) oracle is not required.

We consider the experiment with two phases:

Phase 1 The adversary can call the following oracles: O_{CrU} (create user), O_{CoU} (corrupt user), O_{CC} (create certonym), and O_S (sign message), which are defined in section 5.1.

Phase 2 The adversary outputs some data.

The adversary wins the experiment if

- Case 1: The data is a certonym C such that C is not the output of O_{CC}; or
- Case 2: The data is a triple (C, s, m) such that $\mathsf{Verif}(C, s, m)$ outputs Accept, and C, m is not the input to O_S, and for all users u, if $\text{owner}(u, C)$ then u has not been provided as an input to O_{CoU}.

Definition 4. *(certonym owner). owner(u, C) holds if O_{CC} has been called with argument u and output C.*

We denote the fact that the adversary wins the above game by $\mathsf{Exp}^{\text{Unforge}}_{\text{cert},\mathcal{A}}(\lambda) = 1$.

Definition 5. *(Unforgeability). Given a security parameter λ, it is said that a certonym scheme holds the property of unforgeability, if for any polynomial-time $\mathcal{A}$, the probability of $\mathcal{A}$ winning the unforgeability experiment is as follows*

$$Succ^{Unforge}_{cert}(\mathcal{A}(\lambda)) = Pr[\mathsf{Exp}^{Unforge}_{cert,\mathcal{A}}(\lambda) = 1] \leq negl(\lambda).$$

In parallel of running $\mathsf{Exp}^{\text{Unforge}}_{\text{cert},\mathcal{A}}$, $\mathcal{S}$ is also involved in other two experiments, denoted by $\mathsf{Exp}^{\text{EU-CMA}}_{u}$ and $\mathsf{Exp}^{\text{EU-CMA}}_{Issuer}$. $\mathsf{Exp}^{\text{EU-CMA}}_{u}$ is associated with the user's signatures and $\mathsf{Exp}^{\text{EU-CMA}}_{Issuer}$ is associated with the Issuer's signatures. In either of these two experiments, $\mathcal{S}$ acts as an adversary, aiming to create a forgery of the user's signature or the Issuer' signature. Now we give a formal definition of EU-CMA:

Definition 6. *(Existential unforgeability under adaptive chosen-message attacks (EU-CMA)). This property is defined by using an experiment $\mathsf{Exp}^{EU-CMA}_{sig}$, which is run between a polynomial-time adversary $\mathcal{A}$ and a simulator $\mathcal{S}$. Suppose that $\mathcal{A}$ has access to a public verification key of a digital signature scheme, they can make multiple queries to a signing oracle with messages at the adversary's choice. The query is answered by $\mathcal{S}$. Then $\mathcal{A}$ provides a message and signature pair (m, σ) in which the message was not previously submitted to the signing oracle. A signature scheme is considered to be EU-CMA secure if the probability of the aforementioned adversary succeeding is negligible.*

Theorem 4. *For a suitable security parameter, λ, the certonym scheme is unforgeable if:*

- *Issuer's signature scheme is EU-CMA secure.*
- *A certonym's underlying signature (i.e., a user's signature) is EU-CMA secure.*
- *The function # is a cryptographic hash function and it is considered as a random oracle.*
- *The knowledge proof π is an Non-Interactive Zero-Knowledge Proof (NIZKP).*

Due to space limitation, the proof of this theorem is given in Appendix G[5].

6 Implementation

6.1 Zero-knowledge proofs and encryption

6.1.1 SNARKs. We require a general purpose zero-knowledge proving system and use SNARKs, in part for their short proof sizes which is appropriate for publishing to a blockchain ledger. We use Groth16 [11], instantiated with KZG commitments and implemented with the Gnark software package [2]. We instantiate Groth16 with BN254 as the proving curve. As an implication, our SNARK circuits operate on arithmetic over the scalar field of BN254.

6.1.2 ElGamal encryption. We prove statements about encrypted values. We use ElGamal encryption over an elliptic curve group. We also make use of hashed ElGamal, as discussed later in this section, which is useful when the messages to encrypt are not elements of the elliptic curve group. Proving statements about hashed ElGamal ciphertexts is more expensive than proving the same statements about non-hashed ElGamal ciphertexts, so we use the non-hashed variety when we can.

[5]This appendix is only in the long version of the paper, available on the authors' websites.

6.1.3 2-chains of elliptic curves. A *2-chain* is a pair of elliptic curves such that the base field of one curve (called the 'inner' curve) is the scalar field of the other ('outer') one. In our protocol, when proving statements about encrypted values, our circuits must operate on inputs that are ElGamal ciphertexts. These ciphertexts consist of elliptic curve group elements over some base field. To avoid the need to emulate arithmetic over one field while working within another field, it is important to use an elliptic curve with base field equivalent to the scalar field of the proving curve BN254. In our case, we use the Baby Jubjub curve, which has been designed for this purpose: in particular, BN254 and Baby Jubjub form a 2-chain of curves.

Unfortunately, our use of this 2-chain does not allow us to fully avoid emulated arithmetic. One of our zero-knowledge proofs π_{enc} requires performing operations over scalar field elements of Baby Jubjub, the inner curve of the 2-chain. As such, we must simulate arithmetic in Baby Jubjub's scalar field while working over circuits that operate in its base field. This emulation incurs a penalty in terms of the number of constraints, but is acceptable when the decryption threshold and the number of trustees are not too large.

6.1.4 Hashed ElGamal. Hashed ElGamal is a variant ElGamal scheme in which an encryption E for message m is formed as:

$$E = (g^r, m + \#(PK^r)).$$

An advantage of hashed ElGamal is that the messages need not be group elements and can be arbitrary bytes. In particular, the + can be interpreted as a bitwise operation or as addition over a scalar field. Our proof π_{enc} involves encryptions of scalar values rather than group elements and thus applies hashed ElGamal. In our case, we interpret the addition on the right hand side of the ciphertext as addition over the scalar field of BN254. (It would suffice to use addition over the scalar field of Baby Jubjub, considering that our messages live in the scalar field of Baby Jubjub. However, performing arithemetic over the larger scalar field of BN254 avoids further emulated arithmetic.)

6.1.5 Hashing to elliptic curves and the function ν. Step 3 of the Issue protocol involves the selection of $MAX_q + 3$ pseudorandom elliptic curve group elements. We use group elements, rather than raw bytes, so that these values can be encrypted using elliptic curve ElGamal, in which the message space is group elements. Selecting pseudorandom elements in this way avoids hashed ElGamal and emulated arithmetic. The elements must be chosen in such a way that the next element can be derived from the previous one but the reverse direction is infeasible to compute. Recall that we use the function ν to obtain the next group element from the previous group element, as $\nu(h) = g^{\#(h)}$ (see Section 2.1). This is in effect a naive hash-to-curve protocol. Often, this construction is insecure because the hash function range and the group order are mismatched, causing a non-uniform output. However, use a MiMC-based hash function with range size $|g|$ to ensure uniformity. We omit details related to domain separation and constant time operations.

6.1.6 Parameters. Global parameters are given in Table 1. We use for g a generator of the Baby Jubjub elliptic curve group. Users of the system may select how many trustees to encrypt to and a decryption threshold or may be required to make certain such choices. Table 2 shows how the number of trustees and decryption threshold affect relevant proving performance. Tables 3, 4, and 5 highlight performance impact of the critical system parameters MAX_q and MAX_c. As MAX_q gets larger, the Issue protocol takes longer and the proof π_{issue} increases in complexity, while the protocol to establish a certonym and π_{est} are relatively unaffected. The purpose of MAX_c is to limit the number of certonyms that Alice can create per generation (between BR queries), so that BR query execution remains tractable for the relying party. The SNARK proving times are relatively constant with respect to this parameters, although protocol blind-regroup becomes more computationally intensive for the relying party as MAX_c grows.

6.2 Ledger and smart contracts

We sketch here a smart contract structure for integration of the protocol with a blockchain. We assume that the blockchain supports smart contracts capable of verifying Groth16-based SNARKs that use BN254 as the proving curve; this is the case for Ethereum. A protocol set-up contract Master may help organise protocol parameters and participants. It is here that trustees, the relying party, and the Issuer may publish public keys. The universe of trustees may also be managed here: some (possibly decentralised) authority must take judgements as to which trustees may participate in the protocol. This contract may also be used to publish certonyms or to link them to existing forms of identity. A second contract Bulletin serves as a ledger, containing decryption requests and trustee responses. A third contract will maintain a Merkle tree T.

6.3 Zero-knowledge proof performance

We have conducted experiments to show the time to compute the proofs in our paper (π_{enc}, π_{issue}, π_{est} and π_{fup}), using various values for the parameters involved. Tables showing the results are given in Appendix A.

7 Conclusions

Any practical scheme for the management of digital identities must strike an acceptable balance between providing privacy by default to users and providing the ability for authorities to make queries that are crucial to system regulation or the investigation of criminal activity. Users should be able to see what queries have been made, and hold the querying authorities accountable. We use the term *certonym* for credentials that satisfy these properties that balance user privacy and investigative queries.

Blind-regroup is a query which, given a credential, allows authorities to discover all the credentials created by the user of that credential. Blind-regroup is privacy-preserving, because (unlike a tracing query) it does not reveal the ground identity of the user. Also, it allows the user to continue creating credentials since credentials created after a blind-regroup query are not compromised by the query. Blind-regroup is an essential part of our concept of certonymity.

References

[1] [n. d.]. Certificate transparency. www.certificate-transparency.org.

[2] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. 2024. *Consensys/gnark: v0.11.0.* doi:10.5281/zenodo.5819104

[3] Jan Camenisch and Anna Lysyanskaya. 2004. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*. 56–72.

[4] Jan Camenisch, Ueli Maurer, and Markus Stadler. 1997. Digital payment systems with passive anonymity-revoking trustees. *Journal of Computer Security* 5, 1 (1997), 69–89.

[5] David Chaum. 1983. Blind Signature System.. In *Crypto*, Vol. 83. Springer, 153.

[6] Vanesa Daza, Javier Herranz, Paz Morillo, and Carla Rafols. 2007. CCA2-secure threshold broadcast encryption with shorter ciphertexts. In *International Conference on Provable Security*. Springer, 35–50.

[7] Cécile Delerablée and David Pointcheval. 2008. Dynamic threshold public-key encryption. In *Annual International Cryptology Conference*. Springer, 317–334.

[8] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. 2024. Threshold encryption with silent setup. In *Annual International Cryptology Conference*. Springer, 352–386.

[9] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20 (2007), 51–83.

[10] Hossein Ghodosi, Josef Pieprzyk, and Rei Safavi-Naini. 1996. Dynamic threshold cryptosystems. In *Proceedings of PRAGOCRYPT*, Vol. 96. Citeseer, 370–379.

[11] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 305–326.

[12] Au Man Ho, Susilo Willy, and Mu Yi. 2006. Constant-size dynamic k-TAA. In *Security and Cryptography for Networks, SCN*. 111–125.

[13] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 177–194.

[14] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. 2004. Traceable signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 571–589.

[15] Joon Sik Kim, Kwangsu Lee, Jong Hwan Park, and Hyoseung Kim. 2024. Dynamic Threshold Key Encapsulation with a Transparent Setup. *Cryptology ePrint Archive* (2024).

[16] B. Laurie, A. Langley, and E. Kasper. 2013. Certificate Transparency. RFC 6962 (Experimental).

[17] Benoît Libert. 2024. Simulation-Extractable KZG Polynomial Commitments and Applications to HyperPlonk. In *IACR International Conference on Public-Key Cryptography*. Springer, 68–98.

[18] David Pointcheval and Olivier Sanders. 2016. Short randomizable signatures. In *CT-RSA*. 111–126.

[19] Mark D Ryan. 2017. Making decryption accountable. In *Security Protocols XXV: 25th International Workshop, Cambridge, UK, March 20–22, 2017, Revised Selected Papers 25*. Springer, 93–98.

[20] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[21] W3C. 2018. Decentralized identifiers (DIDs) v0.11:data model and syntaxes for decentralized identifiers. https://w3c-ccg.github.io/did-spec/.

A Zero-knowledge proof performance

All experiments on our zero-knowledge proofs were performed on a MacBook Pro with an Apple M3 Pro chip, throttled to 4 CPUs and a 1 GB soft memory limit. See Tables 2, 3, 4, and 5.

Circuit Type	Num. of trustees	Decryption threshold	Constraints	Proving time (s)
π_{enc}, proof for ThreshEnc	3	1	23,084	0.182
	3	2	23,938	0.190
	3	3	24,574	0.194
	9	3	67,402	0.510
	9	6	72,899	0.541
	9	9	78,397	0.589
	30	10	255,781	1.65
	30	20	305,719	2.33
	30	30	354,408	2.35
	90	30	1,007,805	6.02
	90	60	1,352,005	10.2
	90	90	1,696,205	12.7

Table 2: Performance of π_{enc} circuit

Circuit Type	MAX_q	Constraints	Proving time (s)
π_{issue}, proof for set-up	2	31,414	0.189
	8	63,178	0.343
	32	190,234	1.11
	128	698,458	3.97
	512	2,731,354	15.9

Table 3: Performance of π_{issue} circuit

Circuit Type	MAX_c	T_r depth	Constraints	Proving time (s)
π_{est}, proof for certonym creation	100	40	202,749	1.39
	1,000,000	40	202,765	1.41
	100	60	242,862	1.64
	1,000,000	60	242,878	1.69

Table 4: Performance comparison of π_{est} with varying parameters. We fix $n = 9$ trustees with decryption threshold $t = 6$. A T_r depth of 60 roughly corresponds to a scenario in which every human has created $MAX_q \cdot MAX_c$ certonyms for $MAX_q = 512$ and $MAX_c = 1,000,000$.

Circuit Type	MAX_c	Constraints	Proving time (s)
π_{fup}, for some BR queries	100	3,124	0.0263
	1,000,000	3,128	0.0270

Table 5: Performance of π_{fup}, the proof provided by the relying party when making a BR query of type fup**.**

B Threshold decryption key setup

We assume that there are n trustees, each associated with an index $i \in \{1, ..., n\}$ and a (non-threshold) public key PK_i along with the corresponding secret key SK_i. The vector of these trustee public keys is denoted $\boldsymbol{PK}$. In this description, we will assume that an encryptor (Alice) will use all n trustees. In general, there may be a larger universe of $N > n$ trustees and Alice has the freedom to select a subset of n from the universe.

We also work wth ElGamal encryption and assume that each ciphertext C may be parsed as a pair (c, c') of the form $(g^r, m \cdot PK^r)$.

B.1 Threshold key generation

This section explains how Alice creates a public key TPK and the corresponding secret keys $sk_1, \ldots, sk_n$ such that any $t \leq n$ of them can decrypt information that has been encrypted with TPK. This idea follows Shamir's secret sharing scheme [20].

- *Input*: Number n of trustees, threshold $t \leq n$, and trustee public keys $\boldsymbol{PK} = (PK_1, \ldots, PK_n)$.
- *Method*:

- **Public Inputs**:
  - $\boldsymbol{E} = (e_1, ..., e_n)$
  - TPK
  - $\boldsymbol{PK} = (PK_1, ..., PK_n)$
- **Private Inputs**:
  - $a_0, ..., a_{t-1}$
  - $sk_1, ..., sk_n$
  - randomnesses associated with the encryptions within $\boldsymbol{E}$

Constraints

- sk_i equals $p(i)$, where p is the degree $t-1$ polynomial given by $a_0 + a_1 \cdot x + ... + a_{t-1} \cdot x^{t-1}$
- e_i is a valid encryption (using generator g) to public key PK_i of sk_i, using the given randomness
- $g^{a_0} = TPK$

Figure 6: Proof π_{enc} of correct setup of threshold public key. The proof ensures that the public key was formed to respect the correct decryption threshold. In our implementation, the sk_i are elements of the Baby Jubjub scalar field rather than group elements; as such, the e_i are hashed ElGamal ciphertexts. A new circuit must be compiled for each desired value of t.

(1) Generate a random polynomial of degree $t-1$:

$$p(x) = a_0 + a_1 \cdot x + ... + a_{t-1} \cdot x^{t-1}$$

(2) For $i = 1, ..., n$, compute the evaluations: $sk_i = p(i)$ and create encryption $e_i = \mathsf{Enc}_{PK_i}(sk_i)$ of sk_i for the ith trustee. Let $\boldsymbol{E} = (e_1 \ldots, e_n)$.
(3) Let $TPK = g^{a_0}$.
(4) Constructs a zero-knowledge proof π that all this has been done correctly, described in Figure 6.

- *Output*: $(\boldsymbol{E}, TPK, \pi)$

B.2 Definition of $\mathsf{PartDec}_{sk_i}(C)$

Trustee i parses ciphertext C as $C = (c, c') = (g^r, M \cdot TPK^r)$. To execute $\mathsf{PartDec}_{sk_i}(C)$, the trustee computes $m_i = c^{sk_i}$. Note that Trustee i is given e_i and computes $sk_i = Dec_{SK_i}(e_i)$.

B.3 Definition of $\mathsf{Interpolate}((m_i)_{i \in I})$

(1) The relying party computes $d = \prod_{i \in I} m_i^{\lambda_{0,i}}$ where the Lagrange coefficients $\lambda_{0,i}$ for reconstructing the constant term of a polynomial with t parties are given by: $\lambda_{0,i} = \prod_{\substack{j \in I \\ j \neq i}} \frac{j}{j-i}$

We note that if each trustee has correctly provided PartDec, then

$$d = \prod_{i \in I} c^{p(i) \cdot \lambda_{0,i}} = c^{\sum_{i \in I} p(i) \cdot \lambda_{0,i}} = c^{a_0}$$

(2) The relying party finally computes $M = c' \cdot d^{-1}$, which is the plaintext of C.

C Simulation of the modified KZG scheme

We now discuss how the modified KZG scheme is simulatable. We first recall the KZG scheme [13] and its modification in [17]. Given a security parameter λ, choose asymmetric bilinear groups $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ of prime order $p > 2^{l(\lambda)}$, where $l(\lambda)$ is a polynomial of λ. Let $\alpha \in \mathbb{Z}_p$, $g \in \mathbb{G}$, $g_i = g^{\alpha^i} \in \mathbb{G}$ for $i = (1, ..., d)$, $\hat{g} \in \hat{\mathbb{G}}$, $\hat{g}_1 = \hat{g}^{\alpha} \in \hat{\mathbb{G}}$ and $\hat{g}_2 = \hat{g}^{\alpha^2} \in \hat{\mathbb{G}}$. The public parameter is $(g, \{g_i\}_{i=1}^d, \hat{g}, \hat{g}_1, \hat{g}_2)$. Note that the secret parameter α is not used later. The target is to prove a polynomial

$$f(x) = \sum_{i=0}^{deg(f)} f_i x^i,$$

where the degree of the polynomial $deg(f)$ is up to d. The coefficients f_i are known to the prover but not the verifier.

In the original KZG scheme, a commitment to this polynomial is

$$C = g^{f(x)} = \prod_{i=0}^{deg(f)} g_i^{f_i}.$$

The commitment can be opened by outputing $f(x)$. Given an input z, a witness of such a commitment is

$$w_z = g^{\frac{f[\alpha] - f(z)}{\alpha - z}}.$$

Note that $(x - z)$ must perfectly divide the polynomial $f(x) - f(z)$. This witness can be verified by checking the pairing equation

$$e(C, \hat{g}) = e(w_z, \hat{g}_1 / \hat{g}^z) \cdot e(g, \hat{g})^{f(z)}.$$

The original KZG paper makes use of a Type I pairing, i.e. $\mathbb{G} = \hat{\mathbb{G}}$.

A randomized version of KZG in Material C of [17] works as follows. In this modified KZG scheme, a commitment to a polynomial $f(x) = \sum_{i=0}^{deg(f)} f_i x^i$ is obtained by choosing $\gamma \leftarrow_R \mathbb{Z}_p$ and computing

$$C = g^{\gamma} \cdot \prod_{i=1}^{deg(f)+1} g_i^{f_{i-1}}.$$

Let $F(x) = \gamma + x \cdot f(x)$, $C \cdot g^{-\gamma}$ is a commitment to a polynomial $F_0(x) = F(x) - \gamma = x \cdot f(x)$ for which, given an input z, $F_0(z) = z \cdot y$ and $y = f(z)$.

Assume that $(x - z)$ perfectly divides the polynomial $f(x) - y$ for $z \in \mathbb{Z}_p$. Note that this condition is required in KZG as well. We have

$$q(x) = \frac{F_0(x) - y \cdot x}{x \cdot (x - z)} = \sum_{i=0}^{deg(f)-1} q_i \cdot x^i,$$

and then the witness

$$\pi = \prod_{i=0}^{deg(f)-1} g_i^{q_i} = g^{q(\alpha)} \in \mathbb{G},$$

which is w_z in the original KZG scheme. The following equation holds:

$$e(C \cdot g_1^{-y}, \hat{g}) = e(g, \hat{g})^{\gamma} \cdot e(\pi, \hat{g}_2 \cdot \hat{g}_1^{-z}).$$

Generate a NIZK proof of knowledge of (γ, π). Choose $r_\gamma \leftarrow_R \mathbb{Z}_p$, $R_\pi \leftarrow_R \mathbb{G}$ and compute

$$R = e(g, \hat{g})^{r_\gamma} \cdot e(R_\pi, \hat{g}_2 \cdot \hat{g}_1^{-z}).$$

[linecolor=blue!40,linewidth=1pt]
- **Public Inputs**:
  - Com
  - $nonce_{Issuer}$
  - V_{setup}
- **Private Inputs**:
  - $nonce_{Alice}$
  - h_{MAX_q+2}

Constraints

- Com is a valid commitment for $nonce_{Alice}$
- $h_{MAX_q+2} = v(\dots v(v(nonce_{Alice}) \cdot v(nonce_{Issuer}))\dots) = v^{MAX_q+2}(v(nonce_{Alice}) \cdot v(nonce_{Issuer}))$
- $V_{setup} = \#(h_{MAX_q+2}, 0)$

Figure 7: Specification of proof π_{issue}. Our implementation uses a hash commitment.

[linecolor=blue!40,linewidth=1pt]
- **Public Inputs**:
  - V'
- **Private Inputs**:
  - h'
  - ϵ

Constraints

- ϵ is a positive integer at most MAX_c
- $V' = \#(h', \epsilon)$

Figure 8: Specification of proof π_{fup}

Compute a challenge $c = H(|b|, C, y, z, R)$, where $|b|$ is a label, and $s_\gamma = r_\gamma + c \cdot \gamma$ and $S_\pi = R_\pi \cdot \pi^c$. Return the proof $\tilde{\pi} = (c, s_\gamma, S_\pi)$. To verify $(C, y, z, \tilde{\pi})$, compute

$$R = e(g, \hat{g})^{S_\gamma} \cdot e(S_\pi, \hat{g}_2 \cdot \hat{g}_1^{-z}) \cdot e(C \cdot g_1^{-y}, \hat{g})^{-c}).$$

Return 1 if $c = H(|b|, C, y, z, R)$ and 0 otherwise.

Note that this modified KZG scheme is simulatable under the random oracle model. Let's see how it works. (c, s_γ) is a Schnorr signature. Recall $\pi = g^{q(\alpha)}$. Let $R_\pi = g^{r_\pi}$ and $S_\pi = g^{s_\pi}$, where $s_\pi = r_\pi + q(\alpha) \cdot c$, so (c, s_π) is another Schnorr signature. It is well-known that a Schnorr signature is simulatable under the random oracle model. To do so, randomly choose c, s_γ and S_π and compute R, then output $\tilde{\pi}$ along with C, y, z and $|b|$. As the computation of R is the same as the verification algorithm, so the output of the simulation is indistinguishable from the original proof.

D Zero-knowledge proof specifications

We provide details of various zero-knowledge proofs. All global parameters specified in Table 1 in Section 3.2 serve as public inputs as needed and are omitted from the following three figures: Figure 6 (proof π_{enc} of correct simulation of threshold public key), Figure 7 (proof π_{issue}), Figure 9 (proof π_{est}), and Figure 8 (proof π_{fup}).

[linecolor=blue!40,linewidth=1pt]
- **Public Inputs**:
  - $H'_{MAX_q-\beta}, H'_{MAX_q-\beta+1}$
  - V
  - T_R
  - y
  - W
  - public inputs of π_{enc}
- **Private Inputs**:
  - bit b
  - $h_{MAX_q-\beta}, h_{MAX_q-\beta+1}, h_{MAX_q-\beta+2}, h_{MAX_q-\beta+3}$
  - $\epsilon, \epsilon', \epsilon'', \epsilon'''$
  - $r'_{MAX_q-\beta}, r'_{MAX_q-\beta+1}$
  - Leaf ℓ_{br}
  - Leaf ℓ_{fup}
  - Leaf ℓ_{est}
  - proof_of_presence(ℓ_{br}, T_R)
  - proof_of_presence(ℓ_{fup}, T_R)
  - proof_of_presence(ℓ_{est}, T_R)
  - private inputs of π_{enc}

Constraints

- $\epsilon, \epsilon', \epsilon''$, and ϵ''' are positive integers at most MAX_c
- $H'_{MAX_q-\beta} = \mathsf{Enc}_{TPK}^{r'_{MAX_q-\beta}}(h_{MAX_q-\beta})$
- $H'_{MAX_q-\beta+1} = \mathsf{Enc}_{TPK}^{r'_{MAX_q-\beta+1}}(h_{MAX_q-\beta+1})$
- $v(h_{MAX_q-\beta}) = h_{MAX_q-\beta+1}$
- $v(h_{MAX_q-\beta+1}) = h_{MAX_q-\beta+2}$
- $V = \#(h_{MAX_q-\beta+1}, \epsilon)$
- $W = \#(h_{MAX_q-\beta+1}, y)$
- The constraints of π_{enc} hold
- b is a bit
- $b = 1$ or:
  - $\ell_{\mathsf{br}} = (\#(h_{MAX_q-\beta+2}, \epsilon'), \mathsf{br})$
  - proof_of_presence(ℓ_{br}, T_R) is valid
- $b = 0$ or:
  - $v(h_{MAX_q-\beta+2}) = h_{MAX_q-\beta+3}$
  - $\ell_{\mathsf{fup}} = (\#(h_{MAX_q-\beta+3}, \epsilon''), \mathsf{fup})$
  - proof_of_presence(ℓ_{fup}, T_R) is valid
  - $\ell_{\mathsf{est}} = (\#(h_{MAX_q-\beta+2}, \epsilon'''), \mathsf{est})$
  - proof_of_presence(ℓ_{est}, T_R) is valid

Figure 9: Specification of proof π_{est}. The first public inputs are (non-hashed) ElGamal encryptions of Baby Jubjub group elements. To avoid complexities associated with recursive proofs, we duplicate the proof π_{enc} naively inside π_{est}. The first private input is a bit to indicate whether the certonym is being established on the basis of a prior blind-regroup query of type br **or of type** fup**; the bit is 1 in the latter case. Note that when $b = 0$, the user may not have leaves ℓ_{fup} or ℓ_{est} but may supply dummy values for them and their proof of presence without compromising the proof. Similarly when $b = 1$, the leaf ℓ_{br} and its proof of presence may be dummy values. Capturing this conditionality within one circuit, rather than using two seperate circuits, is important to prevent information leakage about certonyms.**

E The ledger

E.1 Merkle trees

A Merkle tree is a tree in which every node is labelled with the hash of the labels of its children nodes, and possibly some other values. Suppose a node has n children labelled with hash values $v_1, \ldots, v_n$, and has data d. Then the hash value label of the node is the hash of $v_1, \ldots, v_n, d$. Merkle trees allow efficient proofs that they contain certain data. To prove that a certain data item d is part of a Merkle tree requires an amount of data proportional to the log of the number of nodes of the tree. (This contrasts with hash lists, where the amount is proportional to the number of nodes.)

Example: Figure 10 shows a Merkle tree containing data items $c_1, \ldots, c_6$ stored at the leaf nodes (in this tree, there are no data items stored at non-leaf nodes). Figure 11 shows a larger Merkle tree containing data items $c_1, \ldots, c_{32}$ (again in this case stored only at leaves). To demonstrate that c_{11} is present in the tree, it is sufficient to provide the additional data $c_{12}, h_5, h_{14}, h_{16}, h_{20}$, i.e. one data item per layer of the tree. The recipient of this data can then verify the correctness of the root hash h_{21}. Proving that one Merkle tree extends another can also be done in logarithmic space and time, by providing at most one hash value per layer. For example, to demonstrate that the tree of Figure 11 is an extension of the one in Figure 10, it is sufficient to provide the data h_4, h_{17}, h_{20}. The hash value at the root of the tree is called the root hash (or simply the hash) of the tree.

We give examples of the API calls that we rely on in the paper.

- root() returns $R = h(h(h(c_1, c_2), h(c_3, c_4)), h(c_5, c_6))$ when the ledger has the value shown in Figure 10, and returns $S = h_{21} = h(h_{19}, h_{20}) = \ldots$ when it has the value of Figure 11.
- add(x) creates a new leaf node and adds it to the right of all the existing leaf nodes, and stores the data x there. The hashes stored at non-leaf nodes are updated as needed. The add function reorganises parts of the tree as needed. For example, the sequence add(c_7), …add(c_{32}) transforms the ledger in Figure 10 to the one in Figure 11.
- proof_of_presence(c_{11}, R) returns data items stored in the ledger that are sufficient to prove that c_{11} was stored in the ledger when it had the root R. In this case, the list $c_{12}, h_5, h_{14}, h_{16}, h_{20}$ is returned.
- proof_of_extension(R, S), returns data items stored in the ledger that are sufficient to prove that version of the ledger with root S is an add-only extension of its previous version which had root R. In this case, the list h_4, h_{17}, h_{20} is returned.

F Proof of correctness

F.1 Notation

We use fup-BR (resp. br-BR) to denote a BR query that is requested using a tuple with its second entry set to fup (resp. br).

We also extend our previous notation:

- $\mathsf{Vfrom}_j(h) = \{\#(v^j(h), \epsilon) \mid 1 \leq \epsilon \leq MAX_c\}$
- $\mathsf{Cfrom}_j(h) = \{(G, H, V, W, y, \pi) \in All \mid V \in \mathsf{Vfrom}_j(h)\}$

We use the term *the first generation* to refer to the set of all certonyms that a user can create after the Issue protocol has been run but prior to any BR query.

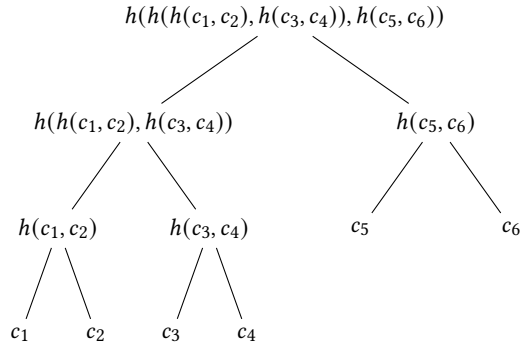


Figure 10: A Merkle tree containing items $c_1, \ldots, c_6$.

F.2 Intermediate Lemmas

Recall that a user Alice creates a chain of values $h_0, ..., h_{MAX_q+2}$ during the Issue protocol. We will refer to these values throughout the lemmas in this section.

Lemma 1. *Any certonym that a user Alice creates must belong to the set* $\mathsf{Cfrom}_0(h)$ *for some* $h \in \{h_1, ..., h_{MAX_q+1}\}$.

Proof. By comparing Step 4 of Figure 3 to Step 8 of Figure 2, it is clear that the first certonym Alice can create must be an element of $\mathsf{Cfrom}_0(h_{MAX_q+1})$. Any subsequently created certonym $C = (G, H, V, W, y, \pi)$ must construct the value V as $\#(h', \epsilon)$ for some h' that Alice knows and that satisfies $v^i(h') = h_{MAX_q+1}$ for some integer $i \leq MAX_q$. The only such possibilities for h' are $h_1, ..., h_{MAX_q}$. Alice cannot construct the V value using h_0 because the construction would require that she know a pre-image of h_0; she cannot know one with non-negligible probability because h_0 is randomly chosen. □

Lemma 2. *For any* h, *if* $\mathsf{Cfrom}_0(h) \neq \emptyset$ *and* $\mathsf{Cfrom}_0(v(h)) = \emptyset$ *then* $h = h_{MAX_q+1}$.

Proof. By Lemma 1, we have that $h \in \{h_1, ..., h_{MAX_q+1}\}$. Let C be an element of $\mathsf{Cfrom}_0(h)$. In order for C to have been established, we have by Figure 3, Step 4 that the Merkle tree contains leaves:

- $(\#(v(h), \epsilon), \mathsf{br})$, or
- $(\#(v^2(h), \epsilon'), \mathsf{fup})$ and $(\#(v(h), \epsilon''), \mathsf{est})$

In the first case, that leaf was either inserted during the Issue protocol (Step 8) or during a BR query with respect to a certonym in the set $\mathsf{Cfrom}_0(v(h))$. The latter case is not possible by the premise that $\mathsf{Cfrom}_0(v(h)) = \emptyset$. As such, the leaf was inserted during the Issue protocol, and we have by Steps 5 and 8 of Figure 2 that $v(h) = h_{MAXq+2}$, which implies that $h = h_{MAXq+1}$. □

In the second case, the leaf $(\#(v(h), \epsilon''), \mathsf{est})$ was either inserted during the Issue protocol (Step 8) or during a certonym establishment of a certonym in the set $\mathsf{Cfrom}_0(v(h))$. The latter case is not possible by the premise that $\mathsf{Cfrom}_0(v(h)) = \emptyset$. As such, the leaf was inserted during the Issue protocol, and we have by Steps 5 and 8 of Figure 2 that $v(h) = h_{MAXq+2}$, which implies that $h = h_{MAXq+1}$.

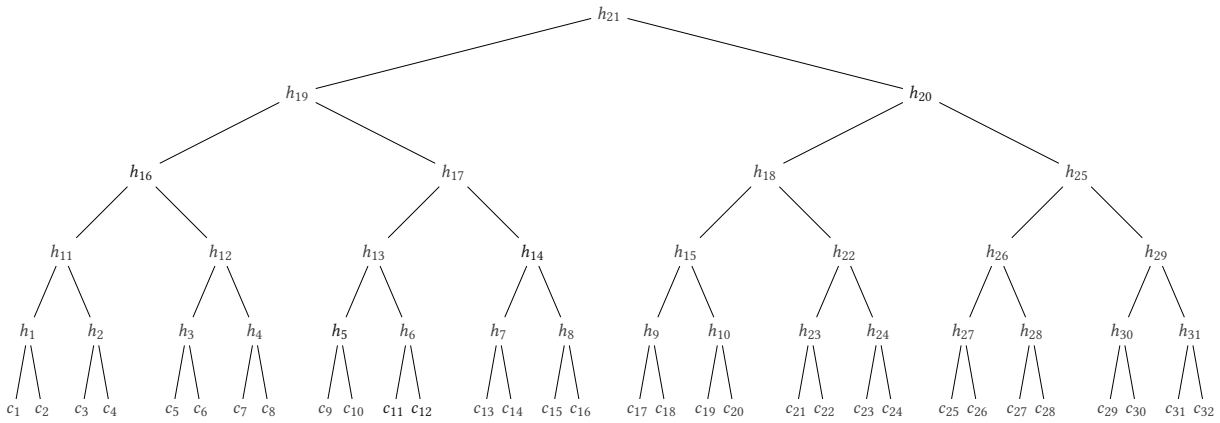


Figure 11: A Merkle tree containing items $c_1, \ldots, c_{32}$. To demonstrate that c_{11} is present in the tree, it is sufficient to provide the additional data $c_{12}, h_5, h_{14}, h_{16}, h_{20}$. To demonstrate that this tree is an extension of the one in the previous figure, it is sufficient to provide the data h_4, h_{17}, h_{20}.

F.3 Proof of executability

We prove Claim 1 by induction. The base case is that no queries have been requested. Alice, to establish a certonym $C \in \mathsf{Cfrom}_0(h_{MAX_q+1})$ in the initial generation, can use the leaf $(\#(h_{MAX_q+2}, 0), \mathrm{br})$, as inserted into the tree during the Issue protocol, in order to complete Step 5 of the Establishment protocol. Our inductive hypothesis is that after i queries, Alice is able to (and does) produce a certonym $C' \in \mathsf{Cfrom}_0(h_{MAX_q-i+1})$. When Alice produces this certonym, $(\#(h_{MAX_q-i+1}, \epsilon), \mathrm{est})$ is added as a leaf to the Merkle tree (for some $\epsilon \leq MAX_c$). We now want to consider whether after the $i+1$-st query, that Alice will be able to produce a certonym $C'' \in \mathsf{Cfrom}_0(h_{MAX_q-i})$.

Case 1. The $i+1$-st query is of type br. This type of query is executed with respect to a certonym that the relying party has not already linked to Alice via any prior BR query. As such, the input certonym is a member of $\mathsf{Cfrom}_0(h_{MAX_q-i+1})$. The query request Merkle tree leaf will be of the form (V, br), where V is of the form $\#(h_{MAX_q-i+1}, \epsilon')$ for some $\epsilon' \leq MAX_c$. This leaf's existence is sufficient for Alice to create a certonym in $\mathsf{Cfrom}_0(h_{MAX_q-i})$, as desired.

Case 2. The $i+1$-st query is of type fup. This type of query will find a certonym from Alice of the latest already-linked generation and the query request leaf will be of the form (V', fup), where V' is of the form $\#(h_{MAX_q-i+2}, \epsilon'')$ for some $\epsilon'' \leq MAX_c$. This leaf, combined with the leaf $(\#(h_{MAX_q-i+1}, \epsilon), \mathrm{est})$ that we already established resides in the Merkle tree, are sufficient for Alice to create a certonym in $\mathsf{Cfrom}_0(h_{MAX_q-i})$, as desired.

F.4 Proof of completeness

We prove Theorem 1.

Proof. We argue three cases separately, which correspond to the three `return` statements in Figure 4.

Case 1. First, we assume that the initial `if` statement is false. In particular, there does not exist any $h^* \in S$ such that $V \in \mathsf{Vfrom}(h^*)$. In this case, a relying party will use the BR query mechanism to decrypt the plaintext of H, which we denote h. The relying party will then compute $\mathsf{Cfrom}(h)$, which is a set containing all of the user's certonyms, except (by Lemma 1) possibly those that belong to $\mathsf{Cfrom}_0(h')$ where $\nu^i(h') = h$ for some $i \geq 1$. For the sake of contradiction, we assume that such a certonym C' has been established. By Lemma 2 we may assume without loss of generality that $i = 1$, namely that $\nu(h') = h$ and by Lemma 1 we may assume that $h' \neq h_{MAX_q+1}$. In order for C' to have been established, we have by Figure 3, Step 4 that the Merkle tree contains leaves:

- $(\#(\nu(h'), \epsilon), \mathrm{br})$, or
- $(\#(\nu^2(h'), \epsilon'), \mathrm{fup})$ and $(\#(\nu(h'), \epsilon''), \mathrm{est})$

We will find that in either scenario we reach a contradiction. In consideration of the first bullet point, using our assumption that $h' \neq h_{MAX_q+1}$, the existence of Merkle tree leaf $(\#(\nu(h'), \epsilon), \mathrm{br})$ implies that a BR query was requested with respect to a certonym belonging to $\mathsf{Cfrom}_0(\nu(h')) = \mathsf{Cfrom}_0(h)$. We then have that $h \in S$, by the fourth line of pseudocode in the final `else` block of Figure 4. This is a contradiction of the initial assumption of Case 1.

Turning to the second bullet point, the existence of those leaves implies that a fup-BR query was requested with respect to a certonym $C'' = (G'', H'', V'', W'', y'', \pi'')$ belonging to $\mathsf{Cfrom}_0(\nu^2(h'))$. The result of such a query is that the plaintext g'' of G'' will be decrypted by the relying party and added to S. By the construction of certonyms, we have that $g'' = \nu(h')$ and we have already justified the assumption that $\nu(h') = h$. As such, we have that $h \in S$, a contradiction of the initial assumption of Case 1.

Case 2. Here we assume that the initial `if` statement is true but the second `if` statement is false. As in Figure 4, we set $h' = \arg\max_{j \in \mathbb{Z}_{\geq 0}} \{\exists h' \in S : \nu^j(h') = h\}$. By the condition in the second `if` statement, we have that $\mathsf{Cfrom}_0(h') = \emptyset$.

The query result will include any certonym of Alice that is an element of $\mathsf{Cfrom}(h')$. For the sake of contradiction, we assume that a certonym C' has been established but is not returned by the query.

By Lemma 2 we may assume that $C' \in \text{Cfrom}_0(h^*)$, where $v(h^*) = h'$. But then we have $C' \in \text{Cfrom}_0(h^*)$ but $\text{Cfrom}_0(v(h^*)) = \emptyset$, which by the same lemma implies that $h^* = h_{MAX_q+1}$. But this is a contradiction because then the certonym C is an element of $\text{Cfrom}_0(h_{MAX_q+2})$, contradicting Lemma 1.

Case 3. We assume that both `if` statements are true. Define h' as in Figure 4 and let C' be an element of $\text{Cfrom}_0(h')$.

The query result will include any certonym of Alice that is an element of $\text{Cfrom}(h'')$, where h'' is such that $v(h'') = h'$. For the sake of contradiction, we assume that a certonym C^* has been established but is not returned by the query.

By Lemma 2 we may assume that $C^* \in \text{Cfrom}_0(h^*)$, where $v(h^*) = h''$. In order for C^* to have been established, we have by Figure 3, Step 4 that the Merkle tree contains leaves:

- $(\#(v(h^*), \epsilon), \text{br})$, or
- $(\#(v^2(h^*), \epsilon'), \text{fup})$ and $(\#(v(h^*), \epsilon''), \text{est})$

We will find that in either scenario we reach a contradiction. In consideration of the first bullet point, we have that the br-BR query implies that $v(h^*) = h'' \in S$, which contradicts the fact that $h' = \arg\max_{j \in \mathbb{Z}_{\geq 0}} \{\exists h' \in S : v^j(h') = h\}$. Turning to the second bullet point, we have that the fup-BR query also implies that $h'' \in S$, which yields the same contradiction as before. □

F.5 Proof of soundness

We prove Theorem 2.

- For a user u, let $h_{0,u}$ be the value created by the user in Step 2d of Figure 2.
- Let $V^{\cap}_{u_1,u_2}$ be a boolean random variable indicating that there exists non-negative integers $\epsilon, \epsilon' \leq MAX_c$, $i \leq MAX_q$, and $j \leq 2MAX_q$ such that $\#(v^i(h_{0,u_1}), \epsilon) = \#(v^j(h_{0,u_2}), \epsilon')$. (Here, the randomness underlying $V^{\cap}_{u_1,u_2}$ is the randomness associated with the selection of h_{0,u_1} and h_{0,u_2}.)

We start by restating a folklore *birthday problem* upper bound:

Lemma 3. *(Folklore, presented without proof.) If n random values are chosen from a space of size R, then the probability that at least two values are the same is at most $\frac{n^2}{2R}$.*

Lemma 4. *Suppose a certonym created by user $u_1 \neq u_2$ is returned as part of a BR query execution done in relation to a certonym of user u_2. Then we have that $V^{\cap}_{u_1,u_2} = 1$.*

Proof. Let h be the plaintext value decrypted with the help of trustees during a BR query request done with respect to a certonym of u_2. The relying party will then compute $\text{Cfrom}(h)$, which by definition involves computation of $v^k(h)$ for $k = 1, ..., MAX_q$. Considering that h may be equal to $v^\ell(h_{0,u_2})$ for any value $\ell \in \{0, ..., MAX_q\}$, we have that the relying party will consider a certonym C with value V as belonging to u_2 exactly when V is in the set $\{\#(v^j(h), \epsilon') \mid j \leq 2MAX_q,\ \epsilon' \leq MAX_c\}$. If a certonym of u_1 with value V' is in this set, then by the definition of certonyms, it must be of the form $\#(v^i(h_{0,u_1}), \epsilon)$ for some $i \leq MAX_q$ and $\epsilon \leq MAX_c$. We then have that $\#(v^i(h_{0,u_1}), \epsilon) = \#(v^j(h_{0,u_2}), \epsilon')$, which means that $V^{\cap}_{u_1,u_2}$. □

Lemma 5. *If users u_1, u_2 do not collude, then*

$$\Pr[V^{\cap}_{u_1,u_2} = 1] \leq O\left(\frac{(MAX_c \cdot MAX_q)^2}{R}\right).$$

Proof.

$$\Pr[V^{\cap}_{u_1,u_2}] \tag{1}$$
$$\leq \Pr[h_{0,u_1} = h_{0,u_2}] + \Pr[V^{\cap}_{u_1,u_2} | h_{0,u_1} \neq h_{0,u_2}] \tag{2}$$
$$\leq \frac{1}{R} + \Pr[V^{\cap}_{u_1,u_2} | h_{0,u_1} \neq h_{0,u_2}] \tag{3}$$
$$\leq \frac{1}{R} + \frac{(2 \cdot MAX_c \cdot (2MAX_q + 1))^2}{2R} \tag{4}$$
$$= O\left(\frac{(MAX_c \cdot MAX_q)^2}{R}\right) \tag{5}$$

The first inequality follow from conditioning on whether $h_{0,u_1} = h_{0,u_2}$ and upper bounding the probabilities $\Pr[h_{0,u_1} \neq h_{0,u_2}]$ and $\Pr[V^{\cap}_{u_1,u_2} | h_{0,u_1} = h_{0,u_2}]$ by 1. The second inequality follows from the definition of the Issue protocol, in which a uniformly random element is jointly selected. The third inequality follows from applying Lemma 3 to the scenario of modelling the hash function (with range of cardinality R) as a random oracle, uniformly at random selecting the elements of the sets $\{\#(v^i(h_{0,u_1}, \epsilon)) : i \leq MAX_q, \epsilon \leq MAX_c\}$ and $\{\#(v^j(h_{0,u_2}, \epsilon')) : j \leq 2MAX_q, \epsilon' \leq MAX_c\}$ to test whether they have a non-empty intersection, and the observation that the total number of selected elements is at most $2 \cdot MAX_c \cdot (2MAX_q)$. □

The proof of Theorem 2 is now an immediate corollary of Lemmas 4 and 5.

G Unforgeability proof

Here is the proof of the unforgeability property defined in Theorem 4.

Proof. Intuitively, to successfully forge a signature on a message m chosen by the adversary using a certonym $C = (G, H, V, \pi)$, there are the following two cases:

Case 1: The adversary obtains a valid certonym C, which an Issuer does not create. This indicates that $\mathcal{A}$ has successfully forged a certonym, which includes Issuer's signature σ_H proved in π and this is a signature from the relying party on a ciphertext with the same plaintext as H. This will contradict the assumption that Issuer's signature scheme is EU-CMA secure.

Case 2: The adversary obtains a triple (C, s, m) and $\text{Verif}(C, s, m)$ outputs Accept, indicating that s is a valid signature on the message m by using the certonym C. However, this signature is not created through a signing query O_S and the user u is honest. This means that $\mathcal{A}$ has successfully forged a user's signature s. This will contradict the assumption that the underlying signature which makes use of a certonym is EU-CMA secure.

While $\mathcal{A}$ has the target of winning the experiment $\text{Exp}^{\text{Unforge}}_{\text{cert},\mathcal{A}}$, $\mathcal{S}$ has the target of either breaking the EU-CMA property of Issuer's Schnorr signature scheme or user u's Schnorr signature scheme. For this purpose, during the run of the $\text{Exp}^{\text{Unforge}}_{\text{cert},\mathcal{A}}$ experiment, $\mathcal{S}$ simultaneously runs an $\text{Exp}^{\text{EU-CMA}}_{Issuer}$ experiment and an $\text{Exp}^{\text{EU-CMA}}_{u}$

experiment with their simulators. In these two experiments, $\mathcal{S}$ plays the role of an adversary.

In the $\mathsf{Exp}^{\text{EU-CMA}}_{Issuer}$ experiment, $\mathcal{S}$ is given the target Schnorr signature public key pk_I, and in the $\mathsf{Exp}^{\text{Unforge}}_{\text{cert},\mathcal{A}}$ experiment, $\mathcal{S}$ shares this key with $\mathcal{A}$ as the Issuer's public key.

Given the above, we now prove that the certonym scheme ensures unforgeability because, in either of these two cases, the adversary's advantage in winning the game is negligible.

All hash functions in the protocol are through a random oracle model, which is run by $\mathcal{S}$. We demonstrate how $\mathcal{S}$ handles the oracle queries, listed in section 5.1.

- O_{CrU} (create user): To create an honest user u, $\mathcal{A}$ sends $O_{CrU}(u)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $u \in U$. If yes, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ runs the *Issue* protocol with $\mathcal{A}$, in which $\mathcal{S}$ plays as an Issuer and $\mathcal{A}$ plays as u. To create a new user u, $\mathcal{S}$ initiates the $\mathsf{Exp}^{\text{EU-CMA}}_{u}$ experiment with its simulator to obtain a given public key, $y = g^x$ (where (y, x) are a public and secret key pair for u: $y = pk_u$ and $x = sk_u$), and $\mathcal{S}$ provides y to $\mathcal{A}$. Note that neither $\mathcal{S}$ nor $\mathcal{A}$ knows the secret key x. To get the Issuer's signature, $\mathcal{S}$ sends a signing query to the simulator of the $\mathsf{Exp}^{\text{EU-CMA}}_{Issuer}$ experiment to obtains σ_{MAX_q}. At the end of the protocol, $\mathcal{A}$ obtains $((h_i, H_i)_{0\le i\le MAX_q+1}, \sigma_{MAX_q}, V_{setup})$. $\mathcal{S}$ records the transcript of this oracle run in U and marks that u is honest.
- O_{CoU} (corrupt user): To corrupt a user u, $\mathcal{A}$ sends $O_{CoU}(u)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $u \in U$ and is marked as honest. If not, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ sends a corrupting query on u to the simulator of the $\mathsf{Exp}^{\text{EU-CMA}}_{u}$ experiment, from which $\mathcal{S}$ receives the corresponding secret key x. $\mathcal{S}$ discloses u's secret key x to $\mathcal{A}$ and marks that u is corrupted.
- O_{CC} (create certonym): To obtain a certonym for an honest user u, $\mathcal{A}$ sends $O_{CC}(u)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $u \in U$ and is marked as honest. If not, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ runs the *Establish* protocol with $\mathcal{A}$. In the protocol, $\mathcal{S}$ handles (1) u's Schnorr signature creation by sending a signing query to the simulator of the $\mathsf{Exp}^{\text{EU-CMA}}_{u}$ experiment, (2) the Issuer's Schnorr signature creation by sending a signing query to the simulator of the $\mathsf{Exp}^{\text{EU-CMA}}_{Issuer}$ experiment, and (3) random oracles. $\mathcal{A}$ acts as a user u, dealing with the hash chain and NIZKP π, but $\mathcal{A}$ does not create the user's Schnorr signature. At the end of the protocol, $\mathcal{S}$ creates a certonym $C = (G, H, V, \pi)$, stores it in U and returns it to $\mathcal{A}$.
- O_S (sign message): To request signing a message m using the certonym C and its corresponding key, $\mathcal{A}$ sends $O_S(C, m)$ to $\mathcal{S}$. $\mathcal{S}$ checks whether $C \in U$ and its corresponding user u is marked as honest. If not, $\mathcal{S}$ rejects this query; otherwise, $\mathcal{S}$ sends a signing query to the simulator of the $\mathsf{Exp}^{\text{EU-CMA}}_{u}$ experiment to obtain a Schnorr signature s on m under the key corresponding to C. $\mathcal{S}$ records this result to U and returns the signature s to $\mathcal{A}$. In this oracle, we do not consider that the adversary randomises its certonym C, as randomisation does not affect the analysis result of unforgeability. If $\mathcal{A}$ submits $O_S(C', m)$ to $\mathcal{S}$, in which C' has been randomised, so $C' \notin U$, Issuer verifies C' and the verification passes, meaning that C' is a valid certonym. Issuer then sends the trustees (which is the adversary) a decryption request; as a return, $\mathcal{S}$ will get the original C.

After an arbitrary number of the above queries, $\mathcal{A}$ decides to complete the first phase (the querying phase) and outputs some data D. As noted, there are two types of D which can indicate that $\mathcal{A}$ wins the experiment $\mathsf{Exp}^{\text{Unforge}}_{\text{cert},\mathcal{A}}$.

Case 1: D involves a valid certonym $C = (G, H, V, \pi)$ but C is not in U (and it is not randomised from any $C \in \mathcal{U}$ either), which means that C is not from an output of any O_{CC} queries. When this happens, $\mathcal{S}$ can extract the underlying Issuer's signature, denoted by $\sigma_{\tilde{H}}$ from π. This is a signature on a ciphertext, denoted by $\tilde{H}$, with the same plaintext as H. This signature is forged by $\mathcal{A}$. $\mathcal{S}$ can then submit a valid forge $(\sigma_{\tilde{H}}, \tilde{H})$ to the simulator of the $\mathsf{Exp}^{\text{EU-CMA}}_{Issuer}$ experiment to win this experiment. However, this result will contradict the assumption that Issuer's signature scheme is EU-CMA secure, so the probability of this case happening must be negligible.

Case 2: D involves a triple (C, s, m), in which s is a valid Schnorr signature on m and can be verified under the public key that is associated with C. It is under the following conditions: $C \in \mathcal{U}$, $(s, m) \notin U$ and the corresponding user $u \in U$ and marked as honest. The signature (s, m) is not created through a signing query O_S and the user u is honest. This means that $\mathcal{A}$ has successfully forged a user's Schnorr signature s on a new message m. $\mathcal{S}$ can then submit a valid forge (s, m) to the simulator of the $\mathsf{Exp}^{\text{EU-CMA}}_{u}$ experiment to win this experiment. This will contradict the assumption that the underlying signature which makes use of a certonym is EU-CMA secure. Therefore, the probability of this case happening must be negligible. Note that, as we discussed before, randomisation on a certonym will not affect this part of analysis, since given a randomised certonym C', $\mathcal{S}$ can retrieve the original C by asking the trustees (in this case, the adversary) to do the decryption. Therefore, a triple (C', s, m) is equivalent to the triple (C, s, m).

As either of these two cases only happens with a negligible probability, the theorem follows. □