SMA^{2}RT: Secret-Metadata Attribute-based Anonymous Rate-limited Tokens

Anna Lysyanskaya, Eileen Nolan

Brown University
{anna,emnolan}@cs.brown.edu

Abstract. In high-volume online services—such as privacy-preserving CAPTCHA bypass or metered paywalls—service providers must filter malicious traffic without compromising user privacy. Anonymous tokens with private metadata (ATPM) address this by embedding a hidden bit into a user's token; for example, indicating whether the user is suspected of being a bot. However, existing ATPM constructions are limited by high communication complexity, requiring a fresh interaction with the issuer for every single token. Furthermore, they lack support for fine-grained policy requirements, preventing service providers from verifying user attributes (such as age or subscription status) without stripping away anonymity.

In this work, we bridge this gap by introducing SMA^{2}RT (Secret-Metadata Attribute-based Anonymous Rate-limited Tokens). For the first time in the hidden-metadata context, our construction supports selective attribute disclosure, thereby bridging the gap between the anonymous credentials and anonymous tokens literatures. Our construction leverages signatures on equivalence classes (SEQ) to achieve an "issue once, spend N times" capability. This allows a user to interact with the issuer only once to obtain a master credential and subsequently derive up to N unlinkable, valid tokens locally, without further online communication. This significantly reduces server load and network latency, making the scheme practical for real-time web applications. Each derived token preserves the issuer's hidden metadata bit and supports selective disclosure of the user's attributes.

1 Introduction

Anonymous authentication tokens are a popular cryptographic mechanism that makes it possible to mitigate the damage from Sybil attacks and similar behavior while respecting user privacy. They are used in iCloud Private Relay [54], Google's Private State Tokens [35], and Cloudflare's Privacy Pass [36], just to name a few widely used commercial products.

In a nutshell, such a token has two security properties: (1) it is unforgeable, meaning that an adversary cannot produce more such tokens that it was issued; and (2) it is unlinkable, or privacy-preserving, meaning that an adversarial issuer cannot tell which token was issued in which instance of the issuing protocol. (It is therefore very similar to blind signatures [51,44,5,53,48].)

The ability to embed a secret metadata bit into these otherwise untraceable tokens is very desirable in practice: it allows to flag suspected bots and Sybils already at the time that a token was issued (when, possibly, more information about the user was available), and do so in a way that the recipients cannot detect (and thus can stay one step behind the learning algorithm that flags them). Awareness of such flagged users leads to mitigation of malicious behavior without catastrophic privacy loss (since only one bit of information is embedded) or usability loss (so that users are not overtaxed with CAPTCHAs, e.g.), and thus leads to a reasonable compromise between security, privacy, and usability [45].

Kreuter, Lepoint, Orrù and Raykova [45] defined and realized anonymous tokens with private metadata bit (ATPM) that allow token issuers precisely that ability. Follow-up work by Benhamouda, Lepoint, Orrù and Raykova [12] and by Chase, Durak and Vaudenay [30] further improved the state of the art by giving stronger and more desirable security definitions and meeting them with better constructions. Baldimtsi, Hanzlik, Nguyen, and Yadav [6] (to appear in ACM CCS 2026) introduced *non-interactive* ATPMs where the issuer and token recipient's interaction was limited to just one message from the issuer to the recipient.

There are, broadly, two flavors of ATPMs: publicly verifiable [12,6] where anyone can verify a token's validity, but only an authorized verifier who has the necessary secret key can recover the metadata bit; and privately verifiable ones [45,30].

ATPMs are not the only flavor of privacy-preserving cryptographic tokens that are getting traction in practice. Anonymous credentials [31,22,23,24], and in particular attribute-based anonymous credentials [17,47,52,7,37,29,42] are at the center of the European Union's and member countries' efforts to create a privacy-preserving digital identity app [1,3,8,2,4].

Attribute-based anonymous credentials provide a richer functionality than plain (no attributes) anonymous authentication tokens. Unforgeability of anonymous credentials is not just one-more unforgeability, but rather stronger: there must exist a knowledge extractor that can trace any showing of an anonymous credential to a particular issue instance [47]. Moreover, the attribute-based feature means that (1) at issue time, the user inputs a set of identity attributes that satisfy some policy; the policy is known to both the user and the issuer, while the attributes are only known to the user; (2) when a credential is shown, both the user and the verifier take as input a predicate P that the user's credential attributes (that correspond to the issue instance traced by the knowledge extractor) must satisfy in order for the verifier to accept.

The rate-limiting property is an additional desirable feature that attribute-based anonymous credentials often offer [20,21,19,9,28]. "Rate-limited" means that the user does not need to be issued each token in a separate interaction with the issuer; instead a wallet of tokens can be issued at once, but only up to N tokens per wallet [20] (or per wallet-time period or similar [21,19]) can be spent. Moreover, it is possible to disallow or disincentivize that the same token be spent in two (or more) different transactions. More precisely, when we say "the

same token," we mean that each token can be uniquely associated with a serial number, specifically by defining an explicit function $\mathsf{SN}(T)$ that, on input a token T outputs this token's serial number. When we say "different transactions," we mean that each transaction is uniquely associated with some context (for example, a timestamp and the nature of a transaction and/or a unique session id associated with it); transactions are different if they are associated with different contexts. Rate-limited anonymous credentials make it possible to de-anonymize the user who spends the same token in two different contexts; thus the user can be disincentivized from engaging in this behavior using some mechanism that's external to the cryptographic scheme.

As this property allows a single instance of the Issue protocol to produce an entire wallet of tokens, it is desirable enough in practice that Apple has initiated a standardization effort [56,55] possibly in order to adapt rate-limited anonymous credentials to use in its iCloud Private Relay or a similar service.

To date, state-of-the-art ATPMs admit neither the attribute-based nor the rate-limited features of anonymous credentials. It is desirable to close this gap: The hidden metadata bit is a more valuable signal in the presence of other attributes. For example, if the tokens are used to filter out trolls who anonymously comment in a local chat group, geographic identity attributes (that attest that the commenter is a local resident) may be used in tandem with the kinds of signals conveyed in hidden metadata; for example, the chat group admin might allow local residents to comment no matter what their metadata bit is, but outsiders can comment only if they are not flagged (e.g. if they passed an AI gate, such as a CAPTCHA, at sign-up time). As for rate-limiting, it offers benefits compared to the state of the art in which each token can only be used once because it reduces the load on the issuer, and yet ensures that each user only has a limited number of tokens. Rate-limiting is also better than an unlimited-use credential: for example, in the local anonymous chat group scenario above, a malicious troll whose identity attributes attest that he is local would only be able to comment a limited number of times.

Our contributions. In this paper, we define and realize Secret-Metadata Attribute-based Anonymous Rate-limited Tokens ($\mathsf{SMA}^2\mathsf{RT}$ tokens); they are ATPMs that are attribute-based and rate-limited.

Our definitional contributions can be thought of as combining the definition of rate-limited attribute-based tokens (such as the recent definition of Chairattana-Apirom, Döttling, Lysyanskaya and Tessaro [28]) with attribute-based credential definition (such as the recent definition of Friedrichs, Harding, Lehmann and Lysyanskaya [42], except without the device-binding aspect) and with the definition of publicly verifiable anonymous tokens with private metadata bit.

A definitional subtlety that was discovered in prior work [30] is how to define the metadata hiding experiment. On the one hand, the challenger must allow the adversary to query for the hidden metadata bit of tokens of its choice. On the other hand, unlinkability requires that the tokens that the adversary produces be unlinkable to the issue sessions in which they were produced. How will the challenger know which token is the challenge one, and which ones are permissible

to reveal the hidden metadata bit for? Prior work [30,12,6] gave a "lunchtime" version of the experiment, in which the challenge token was issued after the adversary was done querying the metadata recovery oracle. In contrast, our definitional approach gives the adversary the freedom to query this oracle at any time. To ensure that the challenger does not reveal the challenge bit, we require that a knowledge extractor be part of the metadata hiding experiment. The extractor will produce the serial numbers of the challenge tokens at the time they are issued, and the challenger will not reveal the metadata bit for tokens with these serial numbers.

In a nutshell, we require the following security properties: (1) unforgeability, which means that there exists an extractor that can predict, at the time that a wallet of tokens are issued, the serial numbers of the resulting N tokens; it is impossible for the adversary to produce a token with a serial number outside of that set, or to alter the attributes or the encoded metadata bit corresponding to a token it obtained; moreover, if a token is spent more than once, it can be traced to the adversarial user to whom it was issued; (2) privacy of the metadata bit, which means that, even though the adversary can engage with the issuer on behalf of any number of users and can query the authorized verifier to recover the metadata bit from tokens obtained via non-challenge credentials, it cannot determine the metadata bit encoded in its challenge credential; (3) unlinkability, which means that there exists a simulator algorithm that can generate tokens in a way that's independent of any honest users' identifying information, including their attributes or anything else specific to this user (other than the metadata bit which is fair game); and the view that an adversarial issuer gets when interacting with honest users is indistinguishable from the view it gets with this simulator; and finally (4) exculpability, which means that an adversary that controls the issuer and has oracle access to honest users cannot produce a double-spent token belonging to (and implicating) an honest user unless this honest user indeed double-spent it.

We then construct a $\mathsf{SMA}^2\mathsf{RT}$ scheme, called **Ches**hi**r**e ("Commit-Hash-Encrypt-Sign-Randomize"), in the random-oracle model from structure-preserving signatures on equivalence classes (SPS-EQ), specifically for the Diffie-Hellman equivalence class [43]; we also use the Dodis-Yampolskiy PRF [38]. Our issuing protocol is round-optimal, i.e. it has a request-response structure.

In the Issue protocol, the user identified by the public key $\mathsf{pk} = g^{\mathsf{sk}}$ (where g is an element of an appropriate group) forms a request for a credential. This request will include a commitment C_{A} to the user's attributes, as well as to two seeds for the PRF, and the user's secret key sk, as well as a proof π_{A} that C_{A} was formed correctly with respect to pk and that the user's attributes satisfy whatever policy the issuer needs them to satisfy. In response, the issuer will send to the user an SPS-EQ signature σ on the vector $m = (g_1, C_{\mathsf{F}}, R, E)$, where g_1 is a special generator used for only the message, C_{F} is (essentially) the commitment to the same values as C_{A}, R is a group element (obtained via a random-oracle query) whose discrete logarithm no one knows, and E is a group element that, together with R, encodes the correct metadata bit b: $E = R^{x_b}$, where (x_0, x_1) is

the secret key needed to both embed the secret metadata bit into the credential provided to the user, and to recover it from a token. (To ensure that only one bit of information is encoded in this wallet, the issuer will provide proof that this vector was formed correctly with respect to the public key; (g^{x_0}, g^{x_1})).

When forming the token, SPS-EQ allows the user to randomize m to obtain an equivalent but unlinkable $m' = (\tilde{g_1}, C'_{\mathsf{F}}, R', C') = ({\tilde{g_1}}^{\alpha}, C_{\mathsf{F}}^{\alpha}, R^{\alpha}, E^{\alpha})$ together with the signature σ' on m'. The user will compute a token's serial number S and double-spending tag D from the PRF seed exactly as suggested by Camenisch, Hohenberger and Lysyanskaya [20], and will include a zero-knowledge proof that the sk and seeds used to compute them correspond to the opening of C'_F; and that the user's identity attributes that correspond to the opening of C'_F satisfy whatever predicate the verifier needs them to satisfy in order to accept the token.

It is counter-intuitive — but fortunate from the efficiency point of view — that the zero-knowledge proofs used in our construction can be just Fiat-Shamir-based proofs derived from standard Σ-protocols for relations between discrete-logarithm representations [49] and need not be straight-line extractable [41]. This is because the steps in the security proof that rely on the unforgeability of the SPS-EQ signature scheme do not require the reduction to be able to extract the witnesses from the proofs: the reductions for those steps go through even without the ability to extract those values.

As a result, both the size of the tokens and the complexity of their issuance and verification are dominated by the cost of proving that a committed value is in the integer range between 0 and $N-1$ (i.e. that the token's serial number is the output of $F_s(i)$ for $0 \leqslant i < N$) [16,46,18,27].

Note that we are not the first to use SPS-EQ to construct ATPMs: they were first used by Baldimtsi, Hanzlik, Nguyen and Yadav [6] (to appear in ACM CCS 2026) to construct non-interactive ATPMs; but those were neither attribute-based nor rate-limited.

2 Security Definitions

Syntax. The $\mathsf{SMA^2RT}$ scheme allows a user to obtain a single "master" credential, or wallet, from an Issuer satisfying a specified policy; the issuer embeds a hidden metadata bit into this credential. From this single credential, the user can locally derive up to N unlinkable tokens to redeem later with a Verifier.

We distinguish two types of verifiers: the Authorized Verifier, who will be able to recover the metadata bit from each token using a secret key formed for this purpose; and a regular Verifier who will be able to use just the Issuer's public key to ascertain the validity of the token.

Notation. We denote the security parameter as $\lambda \in \mathbb{N}$ and assume it is provided in unary 1^{λ} to all algorithms. We use $\perp$ to represent an error or failure. For a finite set S, $s \xleftarrow{\$} S$ denotes that s is sampled uniformly at random from S. We denote $|S|$ as the size of the set S. We use the assignment operator $\leftarrow$ for deterministic assignment and tuple handling. Specifically, we write $(x_1, x_2) \leftarrow T$ to denote parsing or retrieving components x_1, x_2 from a tuple or structure T.

Conversely, we write $T \leftarrow (x_1, x_2)$ to denote assigning or packing the elements (x_1, x_2) into the variable T.

Sets and Spaces. Let $\mathcal{T}$ denote the set of tokens generated by honest users during a security experiment. We denote by A the vector of identity attributes selected by the user for a credential request; it comes from ATTR, the space of valid attributes. We let POL be the space of policies and P be a predicate; the attributes A must satisfy a predicate in P for a token to be redeemed. Let CTX denote the spaces for transaction contexts. We let $N \in \mathcal{N}$ denote the maximum number of unlinkable tokens that can be minted from a single master credential, which is decided upon in the setup phase by all parties. A bit $b \in \{0, 1\}$ represents the hidden metadata embedded by the Issuer.

The syntax of the algorithms comprising a $\mathsf{SMA}^2\mathsf{RT}$ scheme is as follows:

- **Setup and key generation.**
  - $(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda, N)$. The setup algorithm generates the public parameters pp. (For example, in our instantiation, these will include generating a description of the bilinear group setting, as well as sampling group elements that will be used to form multi-base Pedersen commitments; the setup phase will also include the selection of a hash function that will be treated as a random oracle in the security proof.) The parameters will also specify the context space.
  - $(\mathsf{sk_I}, \mathsf{pk_I}) \leftarrow \mathsf{KeyGen_I}(\mathsf{pp})$. Generates the secret and public key pair for the issuer.
  - $(\mathsf{sk_V}, \mathsf{pk_V}) \leftarrow \mathsf{KeyGen_V}(\mathsf{pp})$. Generates a key pair for the authorized verifier. The secret key will be used by the issuer to embed a secret metadata bit in the user's credential, as well as by the authorized verifier to recover the hidden metadata bit. The public key will be used by the credential recipient to ensure that the credential is well-formed. (In principle, $\mathsf{KeyGen_V}$ can be combined with $\mathsf{KeyGen_I}$.)
  - $(\mathsf{sk_U}, \mathsf{pk_U}) \leftarrow \mathsf{KeyGen_U}(\mathsf{pp})$. Generates the secret and public keys for the user.
- **Credential issuance.** Let pol be some Boolean function that a user's attributes A must satisfy in order for the issuer to proceed with issuing a credential to the user associated with public key $\mathsf{pk_U}$. For example, pol may ensure that the attributes in A are consistent with the attributes of some other credential that is external to the system.
  In general, the credential may be issued via an interactive protocol with an arbitrary number of rounds. In this paper, we limit our attention to round-optimal issuance, where the protocol has a request-response structure: first, the user generates a request using $\mathsf{Request}$, and then the issuer responds according to $\mathsf{Response}$:
  - $(\mathsf{state}, \mathsf{req}) \leftarrow \mathsf{Request}(\mathsf{pp}, \mathsf{sk_U}, \mathsf{A}, \mathsf{pol})$ The user inputs the parameters pp, their secret key $\mathsf{sk_U}$, their attributes A and the pol that A must satisfy. The algorithm outputs a protocol message req (for example, containing a commitment to the attributes and a proof that they satisfy the pol) and

local secret state which contains all the information needed for generating the N tokens once the issuer responds to req.
(Note that in our syntax, the user does not take $\mathsf{pk_I}$ and $\mathsf{pk_V}$ as input. This is because our security definitions make sense without them; also, the Cheshire construction does not need them. Since they are unnecessary, we omit them so as not to overcrowd the syntax.)

- $\mathsf{cred} \leftarrow \mathsf{Response}(\mathsf{pp}, \mathsf{sk_I}, \mathsf{pk_V}, \mathsf{sk_V}, \mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$ The issuer takes as input the parameters pp, its secret keys $\mathsf{sk_I}$ and $\mathsf{sk_V}$, the user's public key $\mathsf{pk_U}$ and request req, the hidden metadata bit b to embed in the user's credential, as well the requisite pol that the user's attributes must satisfy, and runs $\mathsf{Response}$ that outputs cred.

- **Token generation and verification.** After obtaining a response cred to its request req, the user can generate and spend tokens. Let ctx be a string that describes the context in which the user needs a token. Let P be the predicate that the user's identity attributes need to satisfy in the given transaction. The values $(\mathsf{ctx}, \mathsf{P})$ are given as input to both the user and the verifier. The algorithms for generating and verifying a token are as follows:
  - $T \leftarrow \mathsf{GenToken}(\mathsf{pp}, \mathsf{state}, \mathsf{cred}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{ctx}, \mathsf{ctr}, \mathsf{P})$ The user runs this algorithm locally to mint a token. Taking the master credential (from cred) and their secret state, it can derive up to N fresh tokens; the counter ctr (where $\mathsf{ctr} \in [0, N)$) directs $\mathsf{GenToken}$ that this is the ctr^{th} token in the wallet. More precisely, $\mathsf{GenToken}$ takes as input the user's state that was saved from when the request for the token was generated; the Issuer's response cred to the request, the context ctx for the transaction, the counter ctr, and the predicate P that the verifier requires the user's attributes to satisfy. (These are attributes that were input to $\mathsf{Request}$ at the time when the user formed the request req to which cred was the response.)
  - $0/1 \leftarrow \mathsf{VerifyToken}(\mathsf{pp}, \mathsf{pk_I}, T, \mathsf{ctx}, \mathsf{P})$. A public verification algorithm that checks the validity of a token and ensures the user's attributes satisfy the specific predicate P provided in the input.
  - $0/1 \leftarrow \mathsf{RecoverMetadata}(\mathsf{pp}, \mathsf{sk_V}, T, \mathsf{ctx})$. The authorized verifier runs this algorithm to extract the hidden metadata bit of the token that was embedded by the Issuer.
  - $\mathsf{SN}(T)$: A deterministic algorithm that, on input a token T, determines its serial number S. The security properties of the scheme will ensure that this serial number corresponds to a unique wallet.
  - $\mathsf{pk} \leftarrow \mathsf{IdentifyUser}(\mathsf{pp}, T_1, \mathsf{ctx}_1, T_2, \mathsf{ctx}_2)$. A public algorithm that, on input tokens T_1 and T_2 with contexts $\mathsf{ctx}_1 \neq \mathsf{ctx}_2$ such that $\mathsf{SN}(T_1) = \mathsf{SN}(T_2)$, outputs a public key pk. From the security point of view, we want pk to belong to the user who issued these tokens; the fact that the tokens' serial numbers collide means that this is the same token spent twice.
  - $0/1 \leftarrow \mathsf{VerifyCred}(\mathsf{pp}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{pk_U}, \mathsf{cred}, \mathsf{req})$. An algorithm that verifies whether the I's response cred to a user's request req is correctly formed.

Next, we define the security properties that a $\mathsf{SMA}^2\mathsf{RT}$ scheme must satisfy.

2.1 Correctness

Intuitively, correctness requires that if all parties, $(\mathsf{User}, \mathsf{Issuer}, \mathsf{Verifier})$, follow the protocol honestly, the resulting tokens will be accepted and the embedded metadata will be recovered correctly. A crucial aspect of our definition is the alignment between the public verification interface and the recovery of the private metadata. As observed in prior work [6,30], maintaining two independent notions of validity (one for public verification and one for reading the hidden bit) can open the door to subtle attacks. For instance, a malicious user might craft a malformed token that passes public verification but causes the metadata extraction to fail or yield an undefined result.

To address this, our correctness definition enforces consistency such that the scheme is only considered correct if a validly generated token simultaneously passes $\mathsf{VerifyToken}$ and yields the correct bit b via $\mathsf{RecoverMetadata}$. In a reasonable implementation, the metadata recovery algorithm will only be run on tokens that have already satisfied the public verification predicate. This unified validity is essential for applications where the entity performing public verification (e.g., a gateway) may be distinct from the entity recovering the metadata.

Formally, we define correctness via the following experiment, which asserts that for any valid keys, attributes, and a policy, the generated token is both valid and decrypts to the bit b chosen by the issuer.

Definition 2.1 (Correctness).
Let $\mathsf{SMA}^2\mathsf{RT} = (\mathsf{Setup}, \mathsf{KeyGen_I}, \mathsf{KeyGen_V}, \mathsf{KeyGen_U}, \mathsf{Request}, \mathsf{Response}, \mathsf{GenToken}, \mathsf{VerifyToken}, \mathsf{RecoverMetadata}, \mathsf{SN}, \mathsf{IdentifyUser}, \mathsf{VerifyCred})$ *be algorithms satisfying the above syntax for the* $\mathsf{SMA}^2\mathsf{RT}$ *token scheme. The scheme* $\mathsf{SMA}^2\mathsf{RT}$ *is* correct *for* ATTR, POL *if for all security parameters* $\lambda \in \mathbb{N}$, *all attribute vectors* $\mathsf{A} \in \mathsf{ATTR}$, *all policies* $\mathsf{pol} \in \mathsf{POL}$, *any* $\mathsf{crs} \in [\mathsf{Setup}(1^\lambda, N)]$, *and any* $\mathsf{ctr} \in N$, *the following experiment* $\mathsf{Exp}^{\mathsf{correct}}_{\mathsf{SMA}^2\mathsf{RT}}(\lambda)$ *always outputs 1:*

$\underline{\mathsf{Exp}^{\mathsf{correct}}_{\mathsf{SMA}^2\mathsf{RT}}(1^\lambda)}$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N)$
$(\mathsf{sk_I}, \mathsf{pk_I}) \leftarrow \mathsf{KeyGen_I}(\mathsf{pp})$
$(\mathsf{sk_V}, \mathsf{pk_V}) \leftarrow \mathsf{KeyGen_V}(\mathsf{pp})$
$(\mathsf{sk_U}, \mathsf{pk_U}) \leftarrow \mathsf{KeyGen_U}(\mathsf{pp})$
$(\mathsf{state}, \mathsf{req}) \leftarrow \mathsf{Request}(\mathsf{pp}, \mathsf{sk_U}, \mathsf{A}, \mathsf{pol})$
$\mathsf{cred} \leftarrow \mathsf{Response}(\mathsf{pp}, \mathsf{sk_I}, \mathsf{pk_V}, \mathsf{sk_V}, \mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$
$T, \mathsf{P} \leftarrow \mathsf{GenToken}(\mathsf{pp}, \mathsf{state}, \mathsf{cred}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{ctx}, \mathsf{ctr}, \mathsf{P})$
return $(\mathsf{VerifyToken}(\mathsf{pp}, \mathsf{pk_I}, T, \mathsf{ctx}, \mathsf{P}) \wedge$
$(\mathsf{RecoverMetadata}(\mathsf{pp}, \mathsf{sk_V}, T, \mathsf{ctx}) == b) \wedge$
$\mathsf{VerifyCred}(\mathsf{pp}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{pk_U}, \mathsf{cred}, \mathsf{req}))$

2.2 Unforgeability

We give a "one-more unforgeability" game-based definition in the setting of anonymous tokens with attributes and hidden metadata. The unforgeability experiment $\mathsf{Exp}^{\mathsf{unforge}}_{\mathsf{SMA^2RT}}(\mathcal{A}, \lambda)$ is detailed in Figure 2.

In this game, the adversary $\mathcal{A}$ takes as input the parameters pp as well as the public keys of the honest issuer and the honest authorized verifier. $\mathcal{A}$ interacts with a set of oracles that emulate the honest participants, including the honest issuer, the honest authorized verifier, and honest users. The $\mathcal{O}_{\mathsf{UKGen}}$ oracle allows the adversary to direct an honest user to generate a key pair; while $\mathcal{O}_{\mathsf{User}}$ allows $\mathcal{A}$ to cause a specified user with public key $\mathsf{pk_U}$ to obtain a credential with attributes A and hidden metadata bit b. $\mathcal{O}_{\mathsf{GenToken}}$ directs the honest user to mint a token and return it to $\mathcal{A}$.

The adversary also interacts with the issuer on behalf of adversarial users: the oracle $\mathcal{O}_{\mathsf{Resp}}$ returns a $\mathsf{credential}$ in response to $\mathcal{A}$'s requests req; the adversary has the freedom to specify the attributes A, hidden metadata bit b, and policy. This oracle employs a knowledge extractor $\mathcal{E}$ that, using whatever privileges access is available to the experiment (such as black-box access to the adversary $\mathcal{A}$, or a trapdoor for the parameters pp or access to the adversary's queries to the random oracle) as well as the issuer's secret keys, can output a way to identify the serial numbers $\mathsf{SN}(T)$ of all the tokens that can be associated with this query to $\mathcal{O}_{\mathsf{Resp}}$. For a token scheme to satisfy the unforgeability property, an efficient algorithm for such an extractor $\mathcal{E}$ must exist.

Finally, the adversary $\mathcal{A}$ that outputs a set of tokens satisfying $\mathsf{VerifyToken}$ can win the game in several ways:

- Credential Forgery (Win 1) : The adversary produces a valid token that cannot be linked to any credential issued by $\mathcal{O}_{\mathsf{Resp}}$.
- Bounded Issuance Violation (Win 2): The adversary spends more tokens than the limit N allows for the number of credentials they obtained.
- Traceability Failure (Win 3): The adversary successfully double-spends (reuses a serial number in different contexts) without the $\mathsf{IdentifyUser}$ algorithm correctly revealing the identity of the user.
- Attribute Violation (Win 4): The adversary generates a valid token for a predicate P using a credential whose attributes do not actually satisfy P.
- Metadata Manipulation (Win 5): As discussed above, the adversary generates a token that verifies correctly, but recovers a metadata bit different from the one embedded by the issuer.

Note that we consider a forgery to be valid if it passes the public verification algorithm $\mathsf{VerifyToken}$, regardless of the outcome of the private metadata bit in $\mathsf{RecoverMetadata}$. Specifically, our game grants the adversary a win (Win 5) if they can "mau" the token such that it verifies publicly but produces an incorrect bit $b' \neq b$ during extraction. This choice yields a stronger security guarantee and is intentional to address the practical reality of token uses. As an example, consider a public gateway that verifies tokens before a private backend processes the metadata. In such architectures, a token that passes the public verification

but carries manipulated metadata would be able to bypass the primary access control layer.

Definition 2.2 (Unforgeability). *The token scheme* $\mathsf{SMA^2RT}$ *satisfies* unforgeability *if there exists a polynomial-time knowledge extractor* $\mathcal{E}$ *algorithm such that for any probabilistic polynomial-time adversary* $\mathcal{A}$*, the probability that the experiment* $\mathsf{Exp}^{\mathsf{unforge}}_{\mathsf{SMA^2RT}}(\mathcal{A}, \lambda)$ *shown in Figure 2 outputs* 1 *is negligible.*

Remark 2.3. Our formulation of the unforgeability property ensures all of the following informal requirements:

1. **Bounded issuance:** After Q successful issuance sessions for a policy pol, an adversary cannot output $k > Q \cdot N$ valid tokens with distinct serial numbers, where $N \in \mathcal{N}$ is the maximum tokens allowed per credential.
2. **Predicate correctness:** An adversary cannot generate a valid token T for a predicate P unless the attributes A used during issuance satisfy P.
3. **Metadata integrity:** An adversary with a credential encoding a hidden bit b cannot produce a valid T' such that $\mathsf{RecoverMetadata}(\mathsf{sk_V}, T', \mathsf{ctx}) = 1 - b$.

Game $\mathsf{Exp}^{\mathsf{privacyMB}}_{\mathsf{SMA^2RT}}(\mathcal{A}, \lambda)$:

$(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda, N)$
$(\mathsf{sk_I}, \mathsf{pk_I}) \leftarrow \mathsf{KeyGen_I}(\mathsf{pp})$
$(\mathsf{sk_V}, \mathsf{pk_V}) \leftarrow \mathsf{KeyGen_V}(\mathsf{pp})$
$\mathcal{R} \leftarrow \emptyset$; $\mathcal{L}^* \leftarrow \emptyset$;
// $\mathcal{A}$ accesses Response, Challenge,
// and RecoverMetadata oracles
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Resp}}, \mathcal{O}_{\mathsf{Chal}}, \mathcal{O}_{\mathsf{RecMet}}}(\mathsf{pp}, \mathsf{pk_I}, \mathsf{pk_V})$
// Adversary wins if bit is correct
return $b^* = \beta$

Oracle $\mathcal{O}_{\mathsf{Resp}}(\mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$:

if $\mathsf{req} \in \mathcal{R}$:
  return $\perp$
$\mathsf{cred} \leftarrow \mathsf{Response}(\mathsf{pp}, \mathsf{sk_I}, \mathsf{pk_V}, \mathsf{sk_V}$
  $\mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$
if $\mathsf{cred} \neq \perp$:
  $\mathcal{R} = \mathcal{R} \cup \mathsf{req}$
return cred

Oracle $\mathcal{O}_{\mathsf{RecMet}}(T, \mathsf{ctx})$:

if $\mathsf{VerifyToken}(\mathsf{pp}, \mathsf{pk_I}, T, \mathsf{ctx}, \mathsf{P}) = 0$
  return $\perp$
$S \leftarrow \mathsf{SN}(T)$
if $S \in \mathcal{L}^*$ // Challenge credential
  return $\perp$
return $\mathsf{RecoverMetadata}(\mathsf{pp}, \mathsf{sk_V}, T, \mathsf{ctx})$

Oracle $\mathcal{O}_{\mathsf{Chal}}(\mathsf{pk_U}, \mathsf{req}^*, \mathsf{pol})$:

if $\mathsf{req}^* \in \mathcal{R}$:
  return $\perp$
$\beta \xleftarrow{\$} \{0, 1\}$
$\mathsf{cred}^* \leftarrow \mathsf{Response}(\mathsf{pp}, \mathsf{sk_I}, \mathsf{pk_V},$
  $\mathsf{sk_V}, \mathsf{pk_U}, \mathsf{req}^*, \beta, \mathsf{pol})$
if $\mathsf{cred}^* \neq \perp$:
  $\mathcal{R} = \mathcal{R} \cup \mathsf{req}^*$
// Extract sn of challenge credential
$\{S_1, ..., S_N\} \leftarrow \mathcal{E}(\mathcal{A}, \mathsf{pp}, \mathsf{sk_I}, \mathsf{sk_V})$
$\mathcal{L}^* \leftarrow \{S_1, ..., S_N\}$
return cred^*

Fig. 1. The privacy of metadata bit experiment for $\mathsf{SMA^2RT}$.

2.3 Privacy of Metadata Bit

Next, we define the privacy of the hidden metadata bit for the $\mathsf{SMA}^2\mathsf{RT}$ scheme via the experiment $\mathsf{Exp}^{\mathsf{privacyMB}}_{\mathsf{SMA}^2\mathsf{RT},\mathcal{A}}(\lambda)$, described in Figure 1. This property captures the requirement that the metadata bit b is semantically hidden from the user (and any other party) who does not possess the verifier's secret key $\mathsf{sk_V}$.

In the experiment, the adversary $\mathcal{A}$ is provided with the system's public keys. It has access to the issuance oracle $\mathcal{O}_{\mathsf{Resp}}$ to obtain credentials for an adversarial user with public key $\mathsf{pk_U}$ in response to the request req; when querying this oracle, $\mathcal{A}$ also explicitly provides the metadata bit b. To challenge the privacy of the scheme, $\mathcal{A}$ queries the challenge oracle $\mathcal{O}_{\mathsf{Chal}}$ with a target request req^*. The challenger flips a coin $\beta \xleftarrow{\$} \{0,1\}$ and returns a credential embedding β. The adversary wins if it can correctly guess β with non-negligible advantage over random guessing.

We additionally give the adversary access to a $\mathcal{O}_{\mathsf{RecMet}}$ oracle that recovers the hidden metadata bit from a token. This strengthens the definition compared to one without such an oracle. However, the adversary must not be able to trivially win by recovering the metadata bit from a token derived from the challenge credential. To prevent this, the $\mathcal{O}_{\mathsf{Chal}}$ oracle uses a knowledge extractor $\mathcal{E}$ that extracts the user's PRF seed z_{U_0} from the proof π_{A} in the challenge request req^*. Since the challenger itself samples z_{I_0} during $\mathsf{Response}$, it can compute the final seed $z_0 = z_{U_0} + z_{I_0}$ and derive all N possible serial numbers for the challenge credential, which are stored in $\mathcal{L}^*$. The $\mathcal{O}_{\mathsf{RecMet}}$ oracle checks whether a queried token's serial number belongs to $\mathcal{L}^*$; if so, the token originates from the challenge credential and the oracle returns $\perp$.

Definition 2.4 (Privacy of metadata bit). *The token scheme* $\mathsf{SMA}^2\mathsf{RT}$ *satisfies* Privacy of Metadata Bit *if there exists a polynomial-time knowledge extractor* $\mathcal{E}$ *algorithm such that for any probabilistic polynomial-time adversary* $\mathcal{A}$*, the probability of the output in the experiment* $\mathsf{Exp}^{\mathsf{privacyMB}}_{\mathsf{SMA}^2\mathsf{RT},\mathcal{A}}(\lambda)$*, shown in Figure 1, is negligible:*

$$\left|\mathsf{Pr}[\mathsf{Exp}^{\mathsf{privacyMB}}_{\mathsf{SMA}^2\mathsf{RT},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leqslant \mathsf{negl}(\lambda)$$

Game $\mathsf{Exp}^{\mathsf{unforge}}_{\mathsf{SMA^2RT}}(\mathcal{A}, \lambda)$:

$(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda, N)$
$(\mathsf{sk_I}, \mathsf{pk_I}) \leftarrow \mathsf{KeyGen_I}(\mathsf{pp})$; $(\mathsf{sk_V}, \mathsf{pk_V}) \leftarrow \mathsf{KeyGen_V}(\mathsf{pp})$
$\mathcal{T} \leftarrow \emptyset$; $\mathcal{U} \leftarrow \emptyset$; $\mathcal{U}_\mathcal{A} \leftarrow \emptyset$; $\mathsf{credID} \leftarrow 0$;
// Initialize set of wallets and issued credentials
$\mathcal{D}_\mathsf{W} \leftarrow \emptyset$; $\mathcal{Q}_\mathsf{Cred} \leftarrow \emptyset$;
$\{(T_i^*, \mathsf{P}_i^*, \mathsf{ctx}_i^*)\}_{i=1}^k \xleftarrow{\$}$
  $\mathcal{A}^{\mathcal{O}_\mathsf{UKGen}, \mathcal{O}_\mathsf{User}, \mathcal{O}_\mathsf{Resp}, \mathcal{O}_\mathsf{GenToken}, \mathcal{O}_\mathsf{RecMet}}(\mathsf{pp}, \mathsf{pk_I}, \mathsf{pk_V})$
// Winning Conditions
for $i = 1$ to k :
  // See if $\mathcal{A}$'s output has honest users' tokens
  if $T_i^* \in \mathcal{T}$: **return** 0
  // All outputted tokens must verify
  if $\mathsf{VerifyToken}(\mathsf{pp}, \mathsf{pk_I}, T_i^*, \mathsf{ctx}_i^*, \mathsf{P}_i^*) = 0$: **return** 0
  // Check for duplicates in $\mathcal{A}$'s output
  if $\exists j < i : T_i^* = T_j^*$: **return** 0
  // Win 1: token is not linked to a credential
  if $(\nexists(\mathsf{cred}, b, \mathsf{SN_{gen}}, \mathsf{A}) \in \mathcal{Q}_\mathsf{Cred}$
    s.t. $\mathsf{SN}(T_i^*) \in \mathsf{SerialNums}(\mathsf{SN_{gen}})$: **return** 1
// Win 2: if more valid tokens than limit
if $k > |\mathcal{Q}_\mathsf{Cred}| \cdot N$: **return** 1
for $i = 1$ to k :
  // Win 3: same serial num, dif. context
  if $\exists j < i : (\mathsf{SN}(T_i^*) = \mathsf{SN}(T_j^*)) \wedge$
    $(\mathsf{ctx}_i^* \neq \mathsf{ctx}_j^*)$:
    $\mathsf{pk_U} \leftarrow \mathsf{IdentifyUser}(\mathsf{pp}, T_i^*, \mathsf{ctx}_i^*, T_j^*, \mathsf{ctx}_j^*)$
    if $\mathsf{pk_U} = \perp \vee \mathsf{pk_U} \notin \mathcal{U}_\mathcal{A}$: **return** 1
    return 0
  else :
    if $\exists(\mathsf{cred}, b, \mathsf{SN_{gen}}, \mathsf{A}) \in \mathcal{Q}_\mathsf{Cred}$:
      // Win 4: Attribute Violation
      if $\mathsf{P}_i^*(\mathsf{A}) = 0$: **return** 1
      $b_i^* \leftarrow \mathsf{RecoverMetadata}(\mathsf{pp}, \mathsf{sk_V}, T_i^*, \mathsf{ctx}_i^*)$
      // Win 5: Metadata Manipulation
      if $b_i^* \neq b$: **return** 1
return 0

Oracle $\mathcal{O}_\mathsf{UKGen}()$:

$(\mathsf{sk_U}, \mathsf{pk_U}) \leftarrow \mathsf{KeyGen_U}(\mathsf{pp})$
$\mathcal{U} \leftarrow \mathcal{U} \cup \{(\mathsf{sk_U}, \mathsf{pk_U})\}$
return $\mathsf{pk_U}$

Oracle $\mathcal{O}_\mathsf{User}(\mathsf{pk_U}, \mathsf{A}, b)$:

if $\mathsf{pk_U} \notin \mathcal{U}$:
  return $\perp$
$(\mathsf{state}, \mathsf{req}) \leftarrow \mathsf{Request}(\mathsf{pp}, \mathsf{sk_U}, \mathsf{A}, \mathsf{pol})$
$b \xleftarrow{\$} \{0, 1\}$
$\mathsf{cred} \leftarrow \mathsf{Response}(\mathsf{pp}, \mathsf{sk_I}, \mathsf{pk_V}, \mathsf{sk_V}, \mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$
$\mathsf{credID} \leftarrow \mathsf{credID} + 1$
$\mathcal{D}_\mathsf{W} \leftarrow \mathcal{D}_\mathsf{W} \cup [\mathsf{credID} : (\mathsf{pk_U}, \mathsf{state}, \mathsf{cred})]$;
return (credID)

Oracle $\mathcal{O}_\mathsf{Resp}(\mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$:

if $\mathsf{pk_U} \notin \mathcal{U}_\mathcal{A}$:
  $\mathcal{U}_\mathcal{A} \leftarrow \mathcal{U}_\mathcal{A} \cup \mathsf{pk_U}$
$\mathsf{cred} \leftarrow \mathsf{Response}(\mathsf{pp}, \mathsf{sk_I}, \mathsf{pk_V}, \mathsf{sk_V}, \mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$
if $\mathsf{cred} \neq \perp$:
  $(\mathsf{SN_{gen}}, \mathsf{A}) \leftarrow \mathcal{E}(\mathcal{A}, \mathsf{req}, \mathsf{pp}, \mathsf{sk_I}, \mathsf{sk_V})$
  $\mathcal{Q}_\mathsf{Cred} \leftarrow \mathcal{Q}_\mathsf{Cred} \cup (\mathsf{cred}, b, \mathsf{SN_{gen}}, \mathsf{A})$
return cred

Oracle $\mathcal{O}_\mathsf{GenToken}(\mathsf{id}, \mathsf{ctx}, \mathsf{ctr}, \mathsf{P})$:

$(\mathsf{pk_U}, \mathsf{state}, \mathsf{cred}) \leftarrow \mathcal{D}_\mathsf{W}[\mathsf{id}]$
$T \leftarrow \mathsf{GenToken}(\mathsf{state_{id}}, \mathsf{cred_{id}}, \mathsf{ctx}, \mathsf{ctr}, \mathsf{P})$
$\mathcal{T} \leftarrow \mathcal{T} \cup T$
return T

Oracle $\mathcal{O}_\mathsf{RecMet}(T, \mathsf{ctx})$:

if $\mathsf{VerifyToken}(\mathsf{pp}, \mathsf{pk_I}, T, \mathsf{ctx}, \mathsf{P}) = 0$
  return $\perp$
return $\mathsf{RecoverMetadata}(\mathsf{pp}, \mathsf{sk_V}, T, \mathsf{ctx})$

Fig. 2. The unforgeability experiment for $\mathsf{SMA^2RT}$.

2.4 Unlinkability

Next, we formalize unlinkability of the $\mathsf{SMA}^2\mathsf{RT}$scheme. Unlike previous works on ATPMs [6,30] that define unlinkability via a game where the adversary links a specific token to one of two potential users (or guesses a permutation of token outputs to many users), we adopt a stronger simulator-based definition. This approach captures the security requirement that a token should leak no information about its origin, specifically the issuance session or the user identity, even to a malicious issuer who controls the issuance keys. Specifically, our definition does not require that all credentials issued by the adversary must be under the same metadata bit.

We detail this via the experiment $\mathsf{Exp}^{\mathsf{unlink}}_{\mathsf{SMA}^2\mathsf{RT},\mathcal{A},\mathcal{S}}(\lambda)$ in Figure 3. In this experiment, the adversary $\mathcal{A}$ acts as a malicious I and V. The adversary first generates its public keys and then attempts to distinguish between two worlds: the real world ($\beta = 0$) and the simulated world ($\beta = 1$). The adversary is given access to oracles $\mathcal{O}_{\mathsf{UKGen}}, \mathcal{O}_{\mathsf{Req}}$ and $\mathcal{O}_{\mathsf{GenToken}}$. If the adversary cannot distinguish between these two worlds with non-negligible advantage, it implies that the distribution of real tokens is computationally indistinguishable from the distribution of simulated tokens. Since the simulated tokens are generated without any link to the user's original request or identity, it follows that the real tokens must also be unlinkable to their issuance history.

Definition 2.5 (Unlinkability). *A token scheme* $\mathsf{SMA}^2\mathsf{RT}$ *satisfies* simulator-based unlinkability *if there exists a PPT simulator* $\mathcal{S} = (\mathcal{S}_{\mathsf{Setup}}, \mathcal{S}_{\mathsf{KeyGen_U}}, \mathcal{S}_{\mathcal{E}-\mathsf{TD}}, \mathcal{S}_{\mathsf{Request}}, \mathcal{S}_{\mathsf{GenToken}})$ *such that for any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the advantage of* $\mathcal{A}$ *in the experiment* $\mathsf{Exp}^{\mathsf{unlink}}_{\mathsf{SMA}^2\mathsf{RT},\mathcal{A},\mathcal{S}}(\lambda)$*, as defined in Figure 3, is negligible:*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{unlink}}_{\mathsf{SMA}^2\mathsf{RT},\mathcal{A},\mathcal{S}}(\lambda) = 1] - \frac{1}{2}\right| \leqslant \mathsf{negl}(\lambda)$$

Remark 2.6. Our unlinkability experiment in Figure 3 does not have a mechanism to direct honest users to double-spend their tokens. To strengthen it to allow for such a mechanism, we require that the simulator have an additional algorithm $\mathcal{S}_{\mathsf{DoubleSpend}}$ that takes as input a token T that was previously output by $\mathcal{S}_{\mathsf{GenToken}}$ together with a new context ctx as well as the public key pk of the user whose token it is, and outputs the token T' that shares the serial number with T. We also add another oracle to the experiment, $\mathcal{O}_{\mathsf{DoubleSpend}}$. If $\beta = 0$, then this oracle will direct the honest user to double-spend the token (in a way that is similar to the oracle in the exculpability game in Figure 4); but if $\beta = 1$, this oracle will invoke $\mathcal{O}_{\mathsf{DoubleSpend}}$. In the sequel, we refer to this enhanced variant of unlinkability as *unlinkability with double-spending*.

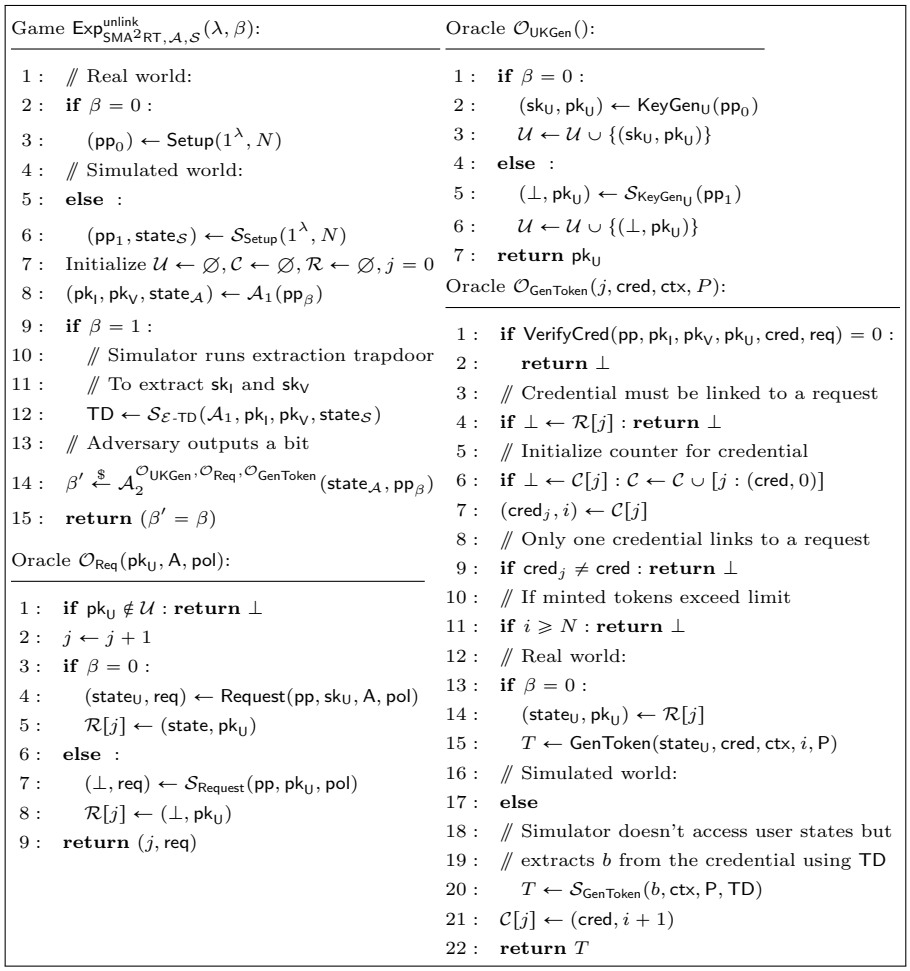


Fig. 3. The unlinkability experiment for $\mathsf{SMA^2RT}$.

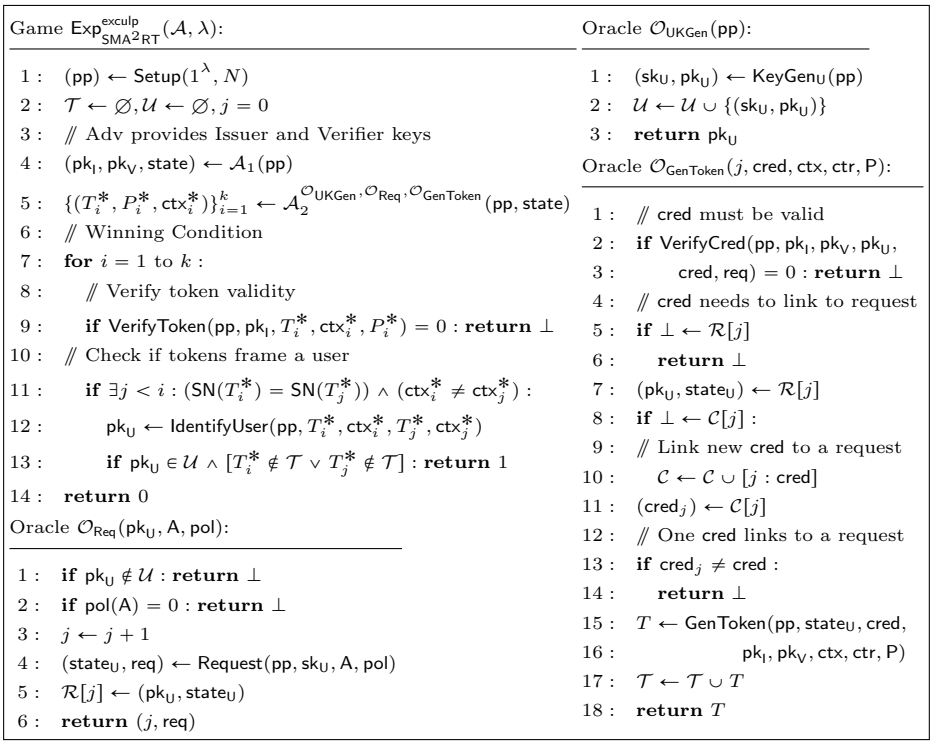


Fig. 4. The exculpability experiment for $\mathsf{SMA^2RT}$.

2.5 Exculpability

We formally define the exculpability of the $\mathsf{SMA^2RT}$scheme via the experiment $\mathsf{Exp}^{\mathsf{exculp}}_{\mathsf{SMA^2RT},\mathcal{A}}(\lambda)$ described in Figure 4.

Intuitively, exculpability ensures that even a malicious issuer (who also acts as the verifier) cannot frame an honest user for misbehavior. In the context of our rate-limited tokens, "misbehavior" corresponds to double-spending (i.e., reusing a serial number across different contexts). The security requirement is that the issuer cannot manufacture a set of valid tokens that trace back to an honest user's identity unless that user actually generated them.

In the experiment, the adversary $\mathcal{A}$ acts as a malicious Issuer and Verifier and generates its own keys. It interacts with honest users via three oracles: $\mathcal{O}_{\mathsf{UKGen}}, \mathcal{O}_{\mathsf{Req}}$, and $\mathcal{O}_{\mathsf{GenToken}}$. The adversary wins the game if it outputs a set of valid tokens that appear to constitute a double-spend (i.e., they share a serial number but differ in context) such that the $\mathsf{IdentifyUser}$ algorithm outputs an honest user $\mathsf{pk_U} \in \mathcal{U}$. This constitutes a successful frame if at least one of the tokens was not actually generated by the honest user (i.e., not in the recorded

set $\mathcal{T}$). This captures two attack scenarios: the adversary either duplicates a legitimate user token into a new context, or fabricates two conflicting tokens entirely from scratch. Both cases prove that the adversary successfully forged "evidence" of a double-spend to frame the user.

Definition 2.7 (Exculpability). *A token scheme* $\mathsf{SMA}^2\mathsf{RT}$ *satisfies exculpability if any PPT adversary* $\mathcal{A}$ *(acting as a malicious Issuer and Verifier) has negligible advantage in the experiment* $\mathsf{Exp}^{\mathsf{exculp}}_{\mathsf{SMA}^2\mathsf{RT}}(\mathcal{A}, \lambda)$*, shown in Figure 4.*

3 Building Blocks for Our Construction

3.1 Signatures on Equivalence Classes (SEQ)

Fuchsbauer, Hanser, and Slamanig [43] introduced a powerful new type of signatures, namely, structure-preserving signatures on equivalence classes. They gave a bilinear-pairing-based construction, and proved it secure in the Type III generic group model. Here, we recall their ideas.

Definition 3.1 (ℓ-Diffie-Hellman message equivalence class). *Let* $\mathbb{G}$ *be a group of prime order* q *(that was generated from a family of groups on input the security parameter* 1^λ*) such that the decisional Diffie-Hellman problem is hard. For* $\ell > 1$*, let* $\mathbf{M} = (M_1, \ldots, M_\ell)$ *and* $\mathbf{M'} = (M'_1, \ldots, M'_\ell)$ *be vectors of elements of* $\mathbb{G}$*.* $\mathbf{M}$ *is Diffie-Hellman equivalent to* $\mathbf{M'}$ *(denoted* $\mathbf{M} \approx_{\mathbb{G}} \mathbf{M'}$*, and in the sequel just called "equivalent") if there exists* $\alpha \in \mathbb{Z}_q$ *such that* $\mathbf{M'} = \mathbf{M}^\alpha = (M_1^\alpha, \ldots, M_\ell^\alpha)$*.*

The Diffie-Hellman equivalence class comes with an algorithm that transforms a message $\mathbf{M}$ into an equivalent message $\mathbf{M'} = \mathbf{M}^\alpha$ given α. Let us call such α the *randomizer* for this equivalence class, and let the corresponding algorithm be $\mathsf{EquivRand}(\mathbf{M}, \alpha) = \mathbf{M}^\alpha$.

Also note that, under the decisional Diffie-Hellman assumption, this equivalence relation is *class-hiding*, i.e. given a random $\mathbf{M}$ and a $\mathbf{M'}$ that is either (1) a random message or (2) a random representative of the same equivalence class as $\mathbf{M}$, it is computationally hard to determine whether it is (1) or (2).

A signature on equivalence classes allows one, given a signature σ on a message m, to transform it into a signature σ' on an equivalent message m'. The original paper of Fuchsbauer, Hanser, and Slamanig (FHS) [43] also cared that the signature be *structure-preserving*, meaning that all elements of the scheme (except the secret key) – its public keys, message spaces, and signatures – be group elements. Here, we do not need the structure-preserving feature, and so will present a relaxed form of the definition, which we call SEQ (rather than SPS-EQ). However, we still use their construction.

Definition 3.2 (Signature on equivalence classes (SEQ)). *Let* $\mathsf{PGen}(1^\lambda)$ *be a parameter generating algorithm, and let* $\approx$ *be an equivalence relation that is well-defined given the parameters* pp *output by* $\mathsf{PGen}(1^\lambda)$*. Let* $\mathsf{EquivRand}$ *be*

the deterministic algorithm that, on input a message m and a random α, outputs $m' \approx m$; if α is random, then m' is a uniformly random representative of the same equivalence class. A signature scheme on $(\mathsf{pp}, \approx)$-equivalence (SEQ) classes consists of the algorithms $(\mathsf{SEQKeyGen}, \mathsf{SEQSign}, \mathsf{SEQVerify}, \mathsf{SEQAdapt})$ *that satisfy the following syntax:*

$\mathsf{SEQKeyGen}(\mathsf{pp})$ *outputs a key pair: the signing key* sk *and the corresponding verification key* vk *of vector length L. It is important that there be an efficient deterministic algorithm that checks whether a given* sk *corresponds to a given* vk.

$\mathsf{SEQSign}(\mathsf{sk}, m)$ *outputs a signature σ on message m.*

$\mathsf{SEQVerify}(\mathsf{vk}, m, \sigma)$ *verifies the signature σ on message m under the verification key* vk.

$\mathsf{SEQAdapt}(\mathsf{vk}, m, \sigma, \alpha)$ *outputs a signature σ' on the message $m' = \mathsf{EquivRand}(m, \alpha)$.*

Moreover, a SEQ is secure if it satisfies correctness, unforgeability and perfect adaptation, defined as follows:

Correctness *(Informally)* $\mathsf{SEQVerify}(\mathsf{vk}, m, \sigma)$ *will always accept a signature σ that was output by* $\mathsf{SEQSign}(\mathsf{sk}, m)$ *or by* $\mathsf{SEQAdapt}(\mathsf{vk}, m', \sigma', \alpha)$ *for* $m = \mathsf{EquivRand}(m', \alpha)$.

Unforgeability *(Informally) Similarly to existential unforgeability under adaptive chosen-message attack of regular signatures, the unforgeability game for SEQ allows a PPT adversary $\mathcal{A}(\mathsf{pp}, \mathsf{vk})$ to query the signer for any number Q of adaptively chosen messages $m_1, \ldots, m_Q$ of its choice. $\mathcal{A}$ wins the game if it can output (m, σ) such that $\forall 1 \leqslant i \leqslant Q$, $m \not\approx m_i$ and yet* $\mathsf{SEQVerify}(\mathsf{vk}, m, \sigma)$.

Perfect adaptation *For all* pp *output by by* $\mathsf{PGen}(1^\lambda)$*, for any (even adversarially chosen)* sk, vk, *m, σ and α such that* sk *corresponds to* vk *and* $\mathsf{SEQVerify}(\mathsf{vk}, m, \sigma)$*, the outputs of the following two algorithms for sampling a signature σ' on the message $m' = \mathsf{EquivRand}(m, \alpha)$ are identically distributed: (1) σ' is the output of* $\mathsf{SEQSign}(\mathsf{sk}, m')$*; (2) σ' is the output of* $\mathsf{SEQAdapt}(\mathsf{vk}, m, \sigma, \alpha)$.

Although the rest of our paper uses the FHL signature on equivalence classes as a black box, we recall its construction just to make sure our paper is self-contained; it is as follows:

$\mathsf{PGen}(1^\lambda)$ returns $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P, \hat{P})$, which is a description of a Type-III bilinear group, consisting of groups $\mathbb{G}_1 = \langle P \rangle, \mathbb{G}_2 = \langle \hat{P} \rangle$ and $\mathbb{G}_T$ of prime order p.

$\mathsf{SEQKeyGen}(\mathsf{pp}, 1^L)$ takes as input the parameters pp but also, in unary, the vector length $L > 1$ that specifies that this scheme is for the L-Diffie-Hellman message equivalence class. It chooses random scalars $x_1, \ldots, x_L \xleftarrow{\$} (\mathbb{Z}_p^*)^L$. The secret key is $\mathsf{sk} \leftarrow (x_1, \ldots, x_L)$, and the public key is $\mathsf{vk} \leftarrow (\hat{X}_1, \ldots \hat{X}_L) = (x_1\hat{P}, \ldots, x_L\hat{P})$. Output $(\mathsf{sk}, \mathsf{vk})$. Assume that all algorithms below have pp as input.

$\mathsf{SEQSign}(\mathsf{sk}, m)$ parses $m = (m_1, \ldots, m_L) \in \mathbb{G}_1^L$. Choose a random scalar $y \xleftarrow{\$} \mathbb{Z}_p^*$ and output $\sigma \leftarrow (Z, Y, \hat{Y})$ with:

$$Z = \sum_{i=1}^{L} x_i \cdot M_i, \quad \text{then sets} \quad Z \leftarrow Z^y, \quad Y = P^{1/y}, \quad \hat{Y} = \hat{P}^{1/y}$$

$\mathsf{SEQVerify}(\mathsf{vk}, m, \sigma)$ parse $\sigma \leftarrow (Z, Y, \hat{Y})$. Output 1 if the following two equations hold:

$$\prod_{i=1}^{L} e(M_i, \hat{X}_i) = e(Z, \hat{Y}) \quad \wedge \quad e(Y, \hat{P}) = e(P, \hat{Y})$$

$\mathsf{SEQAdapt}(\mathsf{vk}, m, \sigma, \alpha)$ parse $\sigma \leftarrow (Z, Y, \hat{Y})$. Choose a fresh random scalar $\psi \leftarrow \mathbb{Z}_p^*$. Output the adapted signature $\sigma' = (Z^{\mu \cdot \psi}, Y^{1/\psi}, \hat{Y}^{1/\psi})$ and corresponding message $m' = (m_1^\mu, \ldots, m_L^\mu)$.

Theorem 3.3 ([43]). *The construction described above is a secure signature scheme on the ℓ-Diffie-Hellman equivalence class in the generic Type-III bilinear group model.*

3.2 Multi-base Pedersen Commitment and a Blindable Variant

Let $\mathbb{G}$ be a prime-order group; let q be its order; suppose $\mathbb{G}$ and q were chosen on input 1^λ such that the discrete logarithm problem in $\mathbb{G}$ is hard. The standard multi-base Pedersen commitment scheme $\mathsf{Ped} = (\mathsf{PedSetup}, \mathsf{PedCom}, \mathsf{PedOpen})$ consists of (1) the setup algorithm $\mathsf{PedSetup}$ that, on input the description of $\mathbb{G}$ and q selects uniformly random generators $y_1, \ldots, y_\ell$ and h and sets $\mathsf{crs}_{\mathsf{com}} = y_1, \ldots, y_\ell, h$; (2) the commitment algorithm PedCom that, on input a vector $\mathbf{x} = (x_1, \ldots, x_\ell)$ chooses uniformly random exponent r from $\mathbb{Z}_q$ and outputs the commitment $C = h^r \prod_{i=1}^{\ell} y_i^{x_i}$; the opening algorithm $\mathsf{PedOpen}$ that, on input a commitment C, a vector $\mathbf{x} = (x_1, \ldots, x_\ell)$ and a value r, checks that $C = h^r \prod_{i=1}^{\ell} y_i^{x_i}$. This commitment scheme is perfectly hiding (C is a uniformly random group element, it is distributed independently of $\mathbf{x}$) and binding under the discrete logarithm assumption (finding $\mathbf{x} \neq \mathbf{x}'$ and exponents r and r' such that $C = h^r \prod_{i=1}^{\ell} y_i^{x_i} = h^{r'} \prod_{i=1}^{\ell} y_i^{x_i'}$ is equivalent to solving the discrete logarithm problem).

In our construction, an issuer will sign (in a way that we will make more clear later) the commitment C for the user. But the user will not present the signed commitment C in the clear; instead, she will produce a blinded version, $C' = C^\alpha$ and provide a zero-knowledge proof of knowledge of α. With a slight modification to the commitment scheme, we can ensure that C' is also binding to $\mathbf{x}$. Consider the following variant, which we will call the *blindable* Pedersen commitment scheme:

Setup $\mathsf{BlindPedSetup}$**:** On input the description of $\mathbb{G}$ and q, select uniformly random generators $y_1, \ldots, y_\ell$, h, and u, and outputs $\mathsf{crs}_{\mathsf{com}} = (y_1, \ldots, y_\ell, h, u)$. (The only difference from standard Pedersen is the addition of the group element u.)

Commitment BlindPedCom**:** On input a vector $\mathbf{x} = (x_1, \ldots, x_\ell)$, choose a uniformly random exponent r from $\mathbb{Z}_q$ and output the commitment $C = uh^r \prod_{i=1}^{\ell} y_i^{x_i}$. (Note: the difference from standard Pedersen is that we multiply the output by the element u.)

Opening BlindPedOpen**:** On input a commitment C, a vector $\mathbf{x} = (x_1, \ldots, x_\ell)$ and values r and α, checks that $C = (uh^r \prod_{i=1}^{\ell} y_i^{x_i})^\alpha$.

It is easy to see that this scheme satisfies the standard hiding and binding properties of a commitment scheme. We will need an additional stronger property, however; it is captured in the following lemma. Intuitively, what the lemma gives us is the guarantee that, after the adversary modifies the commitment C to another commitment $C' = C^\beta$ for some β known to the adversary, it is still committed to the same vector $\mathbf{x}$.

Lemma 3.4. *Let $\mathcal{A}$ be an adversary that, on input $\mathsf{crs}_{\mathsf{com}} = (y_1, \ldots, y_\ell, h, u)$, outputs commitments C and C', their openings $(\mathbf{x}, r, 1)$ and $(\mathbf{x}', r', \alpha)$ and a value β such that $C' = C^\beta$. Then under the discrete logarithm assumption, $(\mathbf{x}', r', \alpha) = (\mathbf{x}, r, \beta)$.*

Proof. Let us construct a reduction that will break the binding property of the standard multi-base Pedersen commitment scheme; in turn, that breaks the discrete logarithm assumption. Our reduction $\mathcal{B}$ takes as input the common random string $(y_1, \ldots, y_\ell, y_{\ell+1}, h)$ for the $\ell+1$-multibase Pedersen commitment scheme, sets $u = y_{\ell+1}$, and sets $\mathsf{crs}_{\mathsf{com}} = (y_1, \ldots, y_\ell, h, u)$. It runs the adversary to obtain $(\mathbf{x}, r, 1)$ and $(\mathbf{x}', r', \alpha)$ and a value β such that $C' = C^\beta$. Suppose $(\mathbf{x}', r', \alpha) \neq (\mathbf{x}, r, \beta)$. Then, if we interpret C' as a standard Pedersen commitment, we have found a way to open it to two distinct values, namely:

$$C' = C^\beta = (uh^r \prod_{i=1}^{\ell} y_i^{x_i})^\beta = h^{r\beta} (\prod_{i=1}^{\ell} y_i^{x_i \beta}) y_{\ell+1}^\beta$$

$$C' = (uh^{r'} \prod_{i=1}^{\ell} y_i^{x_i'})^\alpha = h^{r'\alpha} (\prod_{i=1}^{\ell} y_i^{x_i' \alpha}) y_{\ell+1}^\alpha$$

□

3.3 Dodis-Yampolskiy PRF

Boneh and Boyen [15] introduced the following signature scheme in the bilinear setting where g_1 (resp., g_2) is a generator of the group $\mathbb{G}_1$ (resp. $\mathbb{G}_2$): to sign a message in $m \in \mathbb{Z}_q$ (essentially), compute $\sigma = g_1^{1/(s+m)}$, where s is the signer's secret key, and $\mathsf{vk} = g_2^s$ is the signer's verification key. The verification just checks that $e(\sigma, \mathsf{vk} g_2^m) = e(g_1, g_2)$. Dodis and Yampolskiy [38] showed that this approach can also yield a verifiable random function in the target group under the (sufficiently) decisional strong Diffie-Hellman (SDH) assumption; here, s is the seed of the VRF, $e(g_1, g_2)^{1/(s+m)}$ is the output of the VRF, and σ is the proof that it was evaluated correctly. Subsequent work [20] observed that

without $\mathsf{vk} = g_2^s$, $\sigma = g_1^{1/(s+m)}$ is a secure PRF under SDH in the group $\mathbb{G}_1$ for polynomial-size input domains; it is usually referred to as the Dodis-Yampolskiy PRF, DY for short.

3.4 Σ-Protocols and the Fiat-Shamir Heuristic

In the sequel, we construct protocols to form a blinded Pedersen commitment C, along with a zero-knowledge proof of knowledge of its opening that satisfies a policy or predicate. Additionally, we require proofs that a group element is the output of the DY PRF, where the seed corresponds to an entry of the commitment opening and the input lies within a small integer range. These types of proofs are standard in the literature [40]; we review them here for completeness.

Σ-protocols with perfect simulation. Let $\mathbb{R}$ be an NP relation parameterized by pp. A Σ-protocol for $\mathbb{R}$ consists of four algorithms corresponding to three rounds of communication: first, a prover takes as input the statement x and the witness w and computes an initial message com using the algorithm P_1, then the Verifier responds with a challenge c chosen uniformly at random from some finite space of challenges by running the sampling algorithm Chal, and the Prover computes its response res using the algorithm P_2. Finally, the verifier runs the algorithm Verify on input the parameters pp, the statement x, and the transcript $(\mathsf{com}, c, \mathsf{res})$.

In order to be a secure Σ-protocol with perfect simulation for the relation $\mathbb{R}$, the algorithms $(P_1, \mathsf{Chal}, P_2, \mathsf{Verify})$ must additionally satisfy completeness, perfect zero knowledge, and special soundness. Completeness (informally) means that if the prover and verifier follow the protocol, then the verifier always accepts. Zero-knowledge requires that there exist an efficient simulator algorithm that, on input the statement x and an honest challenge $c \leftarrow \mathsf{Chal}(\mathsf{pp})$, output the rest of the transcript, namely com and res that are distributed identically to the values that the prover would have produced. Special soundness requires that there exist an efficient extractor algorithm that, on input two accepting transcripts for the statement x, $(\mathsf{com}_1, c_1, \mathsf{res}_1)$ and $(\mathsf{com}_2, c_2, \mathsf{res}_2)$ where $\mathsf{com}_1 = \mathsf{com}_2$ but $c_1 \neq c_2$, output a witness w such that $R(x, w)$.

Σ-protocols for proving knowledge of openings of Pedersen commitments are especially well studied. Variations include proofs that openings of two different commitments satisfy a broad class of relations [32,33,26,17,34,47,49], such as equality (e.g. position i in the message committed to in C_1 is equal to position j of the opening of commitment C_2), algebraic relations (e.g. C is a commitment to (x_1, x_2, x_3) where $x_3 = x_1 x_2$) and conjunctions and disjunctions of such relations (e.g., either the opening of C_1 is equal to the opening of C_2, or it is equal to the opening of C_3). Further, it is known how to prove that a committed value belongs to an integer range [16,46,18,27] and that a group element is the output of the DY PRF with a committed seed on input a committed value [20].

From Σ-protocols to NIZK PoK via the Fiat-Shamir heuristic in the random-oracle model. Σ-protocols are rarely used directly; typically they are a building

block towards achieving non-interactive zero-knowledge proofs in the random-oracle model.

Recall that a proof of security in the random-oracle model shows that when an element of a cryptographic construction, such as a hash function, is replaced by a truly random function to which the adversary has oracle access, the resulting scheme satisfies the desired security properties. Although a proof of security in the random-oracle model is not as good a guarantee as a proof of security in the plain model, and many counterexamples have been shown [11], it is considered evidence of security that is good enough in practice.

Recall that a non-interactive zero-knowledge proof of knowledge (NIZK PoK) system [14] for the relation R consists of a Prover algorithm that, on input the public parameters pp, an instance x and a witness w outputs a proof π; and a $\mathsf{Verifier}$ algorithm that, on input pp, an instance x and a proof π, verifies the proof. Such proof system needs to be correct, zero-knowledge, and extractable. Correctness means (informally) that a proof produced by an honest prover is always accepted by the verifier. (Perfect) zero-knowledge means that a simulator that does not have access to the witness (but can program the random oracle) can compute a proof that is distributed identically to a proof produced by the prover (in the random-oracle model). Extractability (in the random-oracle model) means that, given black-box access to an adversary $\mathcal{A}$ that, on input pp, has output a statement x and a proof π accepted by the Verifier, the extractor algorithm can compute the witness w.

Further, for our purposes, it is important that the NIZK PoK be *simulation-extractable*, that is, the extractor can still compute the witness w with black-box access to $\mathcal{A}$, even though as part of its computation $\mathcal{A}$ may request and obtain *simulated* proofs of any statements $x' \neq x$.

The Fiat-Shamir heuristic converts a Σ-protocol to a simulation-extractable NIZK-PoK in the random-oracle model as follows: the prover runs the algorithm for P_1 to obtain com, queries the random oracle on input (x, com) to obtain the challenge c, and then completes the proof by computing res using P_2. Prior work [13,39] established (drawing on the forking lemma and related techniques [50,51,10]) that the resulting proof system is in fact a simulation-extractable NIZK PoK.

Notation. Let R be an NP relation. The notation [25] $\pi \leftarrow \mathsf{PoK}\{w : R(x, w)\}$ denotes that π was computed as the NIZK proof of knowledge of the witness w for the instance x; by default, when using this notation, we mean that it was computed by converting a Σ-protocol we provide for R into an NIZK as described above.

4 Cheshire: Our SMA^{2}RT Token Construction

We now present our construction. The core idea follows a sequence of five steps: first, a user **commits** to their attributes (C_{A}). Next, the issuer uses a **hash** function to derive a random group element (R), which is used to **encrypt** the hidden

metadata bit as E. The issuer adds its own randomness to C_{A}, which we denote as the final commitment C_{F}, and then **signs** the resulting vector (C_{F}, R, E) using a signature on equivalence classes. Finally, the user **randomizes** both the message and the signature to produce the final token. The resulting scheme's acronymic name is **Ches**hi**r**e ("Commit-Hash-Encrypt-Sign-Randomize"), in honor of the Cheshire Cat, a very $\mathsf{SMA}^2\mathsf{RT}$ cat from *Alice in Wonderland*.

Setup. We present the setup phases in Figure 6. The global setup algorithm Setup generates the public parameters pp, which include the description of the bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ of order p, an additional generator g_1, and a common reference string $\mathsf{crs}_{\mathsf{com}}$ for the multi-base blindable Pedersen commitment scheme. Our construction also requires a hash function we denote as $H : \{0,1\}^* \to \mathbb{G}_1$ modeled as a random oracle. We also require another hash function modeled in the random oracle as H_{FS} that is called directly and used for the Fiat-Shamir based zero-knowledge proofs. Additionally, the system-wide parameter N, representing the number of tokens that can be derived from a credential, is fixed.

Following the parameter generation, each party runs their respective key generation algorithm. The U picks a random secret key sk_{U} from $\mathbb{Z}_p$ and computes a corresponding public key $\mathsf{pk} = g^{\mathsf{sk}_{\mathsf{U}}}$. The Issuer runs the key generation for the SEQ signature scheme, SEQKeyGen(pp), as detailed in Section 3. Additionally, the Issuer will produce a zero-knowledge proof of knowledge for the secret signing key. Finally, the AuthorizedVerifier (who may be an external entity separate from the Issuer) generates a key pair for the encryption scheme used to blind the metadata bit, along with a proof of knowledge of the secret key; this key pair will consist of a secret key $(x_0, x_1) \in \mathbb{Z}_p^2$ and the public key (g^{x_0}, g^{x_1}).

Issuance phase. The issuance phase, presented in Figure 7, consists of a single round of communication between a User and an Issuer. The objective is for the User to obtain a credential certifying compliance with a policy pol. This credential effectively functions as a wallet, enabling the User to subsequently mint up to N unlinkable tokens. The protocol initiates with the User constructing a credential request req. Specifically, the User augments their selected attribute set A (of size k) with their secret key and two fresh random seeds, z_{U0} and z_{U1}; these seeds are essential for deriving unique serial numbers during token generation. The User creates a commitment to this vector using the multi-base blindable Pedersen commitment scheme, BlindPedCom, detailed in 3.2. Finally, the User will generate a zero-knowledge proof to attest that the committed values of their attributes vector are valid and satisfy the required pol.

The Issuer begins by verifying the proof provided in the request. To embed the hidden metadata bit b, the Issuer samples a random group element $R = H(\mathsf{req})$ and encrypts the bit using the authorized verifier's secret key in the exponent of R. This produces a ciphertext E, which is generated alongside a proof of correct encryption π_{E}. Subsequently, the Issuer contributes their own random seeds, z_{I0} and z_{I1}, to the user's attribute seeds; this ensures that both the user and the issuer contribute randomness to the final seeds, ensuring that the probability of a collision among the pseudorandom values derived from these seeds is negligible.

Finally, the Issuer computes a SEQ signature on the message tuple $(g_1, C_{\mathsf{F}}, R, E)$. The User receives the complete credential, which comprises the signature, the ciphertext, the issuer's random seeds, the relevant commitments, and the proof of correct encryption.

Token generation and verification. We detail the algorithms for minting and verifying tokens in Figure 8. Upon receiving a credential, the User can mint up to N unlinkable tokens. The process begins with the User executing the public algorithm VerifyCred to validate the credential's integrity. This algorithm performs a series of checks: it verifies that the credential is not empty, confirms the correct computation of the random group element R, validates the SEQ signature and the proof of encryption π_{E}, and ensures that the Issuer's seeds were honestly incorporated into the commitment C_{F}. If successful, VerifyCred returns R, the Issuer's seeds, and the credential, which are used later in the GenToken algorithm. Note that a User only needs to run this algorithm once for every issued credential.

To mint a token, the User increments a counter and derives a serial number S and a double-spending tag D using the Dodis-Yampolskiy PRF (Section 3.3). The PRF keys are derived from the combined seeds of the issuer and the user: $z_0 = z_{\mathsf{U0}} + z_{\mathsf{I0}}$ and $z_1 = z_{\mathsf{U1}} + z_{\mathsf{I1}}$. The user then randomizes the SEQ signature and the underlying message vector (comprising the generator, attribute commitment, and ciphertext) to ensure the resulting token remains unlinkable to the original credential. Finally, a zero-knowledge proof is generated to attest to the ownership of a valid credential, the correct calculation of S and D, the validity of the attribute commitment, and that the counter is within the limit of N tokens.

Verification and recovering the metadata bit. The public verification algorithm VerifyToken allows any party (e.g., a service provider) to check the validity of a token by verifying the randomized SEQ signature and the corresponding proof π_{T} to ensure that the token satisfies some predicate P. In addition to public verification, the AuthorizedVerifier can execute the private algorithm RecoverMetadata. Using its secret key, the AuthorizedVerifier decrypts the randomized ciphertext to extract the hidden metadata bit b.

Double spending identification. Double spending occurs when a User reuses a serial number S for a different transaction context ctx. To execute this, the User must still calculate the double-spending tag D, randomize the Issuer's signature, and generate a valid zero-knowledge proof π_{T} to pass standard verification. However, the system immediately detects the duplicate serial number S.

When detecting collisions across differing contexts, the public IdentifyUser algorithm utilizes the tags D from the conflicting transactions to establish a system of linear equations. Solving this system extracts the user's public key pk_{U}, and identifies the offender. Since this algorithm is public, it creates a strong incentive against double spending. We note that while the algorithm could be designed to output the user's secret key sk_{U}, we consider this an extreme measure for a public mechanism; and follow prior works in revealing only the public identity [6].

Variable	Description
λ	Security parameter
q	Prime order of the group $\mathbb{G}_1, \mathbb{G}_2$
N	Maximum number of tokens allowed per user context
k	Size of user attribute vector $\mathsf{A} = \{a_1, \ldots, a_k\}$
ℓ	Size of final attribute vector $\mathsf{A} = \{a_1, \ldots, a_k, z_0, z_1, \mathsf{sk_U}\}, (\ell = k + 3)$
L	Vector length of SEQ message and $\mathsf{pk_I}$

Fig. 5. Variable names and descriptions for the Cheshire construction.

Algorithm $\mathsf{Setup}(1^\lambda, N)$:

1 : $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g, \hat{g}) \xleftarrow{\$} \mathsf{PPGen}(1^\lambda)$
2 : $\mathsf{crs_{com}} \leftarrow \mathsf{BlindPedSetup}(\mathbb{G}_1, \ell)$
3 : $g_1 \xleftarrow{\$} \mathbb{G}_1$
4 : $\mathsf{pp} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g, \hat{g}), g_1, \mathsf{crs_{com}})$
5 : **return** (pp)

Algorithm $\mathsf{KeyGen_U}(\mathsf{pp})$:

1 : $\mathsf{sk_U} \xleftarrow{\$} \mathbb{Z}_p, \quad \mathsf{pk_U} \leftarrow g^{\mathsf{sk_U}}$
2 : **return** $(\mathsf{sk_U}, \mathsf{pk_U})$

Algorithm $\mathsf{KeyGen_I}(\mathsf{pp})$:

1 : $(\mathsf{sk_I}, \mathsf{pk_I}) \xleftarrow{\$} \mathsf{SEQ.KeyGen}(\mathsf{pp}, L = 4)$
2 : $\pi_\mathsf{I} \leftarrow \mathsf{PoK}\{\mathsf{sk_I} : \mathsf{pk_I} = \hat{g}^{\mathsf{sk_I}}\}$
3 : **return** $(\mathsf{sk_I}, \mathsf{pk_I}, \pi_\mathsf{I})$

Algorithm $\mathsf{KeyGen_V}(\mathsf{pp})$:

1 : $x_0, x_1 \xleftarrow{\$} \mathbb{Z}_p^2$
2 : $\mathsf{pk_V} \leftarrow (g^{x_0}, g^{x_1}), \mathsf{sk_V} \leftarrow (x_0, x_1)$
3 : $\pi_\mathsf{V} \leftarrow \mathsf{PoK}\{(x_0, x_1) : \mathsf{pk_V} = (g^{x_0}, g^{x_1})\}$
4 : **return** $(\mathsf{sk_V}, \mathsf{pk_V}, \pi_\mathsf{V})$

Fig. 6. Setup and key generation for the Cheshire construction.

Algorithm $\mathsf{Request}(\mathsf{pp}, \mathsf{sk_U}, \mathsf{A}, \mathsf{pol})$:

1 : **if** $\mathsf{A} \notin \mathbb{Z}_p^k \vee \mathsf{A}$ doesn't satisfy pol :
2 : **return** $\perp$
3 : // Create user seeds for later PRG use
4 : $z_{\mathsf{U}0}, z_{\mathsf{U}1} \xleftarrow{\$} \mathbb{Z}_p$
5 : // Add $z_{\mathsf{U}0}, z_{\mathsf{U}1}, \mathsf{sk_U}$ to A s.t. $\mathsf{A} \in \mathbb{Z}_p^\ell$
6 : $\mathsf{A.append}\{z_{\mathsf{U}0}, z_{\mathsf{U}1}, \mathsf{sk_U}\}$
7 : $(C_\mathsf{A}, \mathsf{open}_{C_\mathsf{A}}) \xleftarrow{\$} \mathsf{BlindPedCom}(\mathsf{crs_{com}}, \mathsf{A})$
8 : // ZK proof of attributes, policy, and sk
9 : $\pi_\mathsf{A} \leftarrow \mathsf{PoK}\{(\mathsf{A}, \mathsf{open}_{C_\mathsf{A}}) :$
10 : $C_\mathsf{A} = \mathsf{BlindPedCom}(\mathsf{crs_{com}}, \mathsf{A}; \mathsf{open}_{C_\mathsf{A}}) \wedge$
11 : $\mathsf{pol}(\mathsf{A}[1 \ldots k]) = 1 \wedge \mathsf{pk_U} = g^{\mathsf{A}[\ell]}\}$
12 : $\mathsf{req} \leftarrow (\mathsf{pk_U}, C_\mathsf{A}, \pi_\mathsf{A})$
13 : $\mathsf{state} \leftarrow (\mathsf{req}, \mathsf{sk_U}, \mathsf{open}_{C_\mathsf{A}}, z_{\mathsf{U}0}, z_{\mathsf{U}1})$
14 : **return** $(\mathsf{state}, \mathsf{req})$

Algorithm $\mathsf{Response}(\mathsf{pp}, \mathsf{sk_I}, \mathsf{pk_V}, \mathsf{sk_V}, \mathsf{pk_U}, \mathsf{req}, b, \mathsf{pol})$:

1 : $(\mathsf{pk_U}, C_\mathsf{A}, \pi_\mathsf{A}) \leftarrow \mathsf{req}$
2 : **if** $\neg\mathsf{VerifyPoK}(\pi_\mathsf{A})$:
3 : **return** $\perp$
4 : // Create ciphertext for hidden metadata bit
5 : $R \leftarrow H(\mathsf{req}), (x_0, x_1) \leftarrow \mathsf{sk_V}, E \leftarrow R^{(1-b)x_0 + bx_1}$
6 : // Create proof of encryption
7 : $\pi_\mathsf{E} \leftarrow \mathsf{PoK}\{(x_0, x_1, b) : \mathsf{pk_V} = (g^{x_0}, g^{x_1})$
8 : $\wedge\ E = R^{(1-b)x_0 + bx_1}\}$
9 : // Add random contributions to seed attributes
10 : $z_{\mathsf{I}0}, z_{\mathsf{I}1} \xleftarrow{\$} \mathbb{Z}_p$
11 : $C_\mathsf{F} \leftarrow C_\mathsf{A} \cdot y_{\ell-2}^{z_{\mathsf{I}0}} \cdot y_{\ell-1}^{z_{\mathsf{I}1}}$
12 : // Create signature and form credential
13 : $m \leftarrow (g_1, C_\mathsf{F}, R, E)$
14 : $\sigma \leftarrow \mathsf{SEQ.Sign}(\mathsf{sk_I}, m)$
15 : $\mathsf{credential} \leftarrow (\sigma, E)$
16 : **return** $\mathsf{cred} \leftarrow (\mathsf{credential}, C_\mathsf{A}, C_\mathsf{F}, z_{\mathsf{I}0}, z_{\mathsf{I}1}, \pi_\mathsf{E})$

Fig. 7. Credential issuance for the Cheshire construction.

Algorithm $\mathsf{GenToken}(\mathsf{pp}, \mathsf{state}, \mathsf{cred}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{ctx}, \mathsf{ctr}, \mathsf{P})$

$(\mathsf{req}, \mathsf{sk_U}, \mathsf{open}_{C_\mathsf{A}}, z_{\mathsf{U}0}, z_{\mathsf{U}1}) \leftarrow \mathsf{state}$
// On first use verify credential (i.e., $\mathsf{ctr} = 0$)
if $\mathsf{VerifyCred}(\mathsf{pp}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{pk_U}, \mathsf{cred}, \mathsf{req}) = \bot$:
  return $\bot$
else :
$(R, z_{\mathsf{I}0}, z_{\mathsf{I}1}, \mathsf{credential}) \leftarrow \mathsf{VerifyCred}(\mathsf{pp}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{pk_U}, \mathsf{cred}, \mathsf{req})$
$(\sigma, E) \leftarrow \mathsf{credential}$
$(\mathsf{credential}, C_\mathsf{A}, C_\mathsf{F}, z_{\mathsf{I}0}, z_{\mathsf{I}1}, \pi_\mathsf{E}) \leftarrow \mathsf{cred}$
if $\mathsf{ctr} \geqslant N$:
  return $\bot$
// Construct the message
$m \leftarrow (g_1, C_\mathsf{F}, R, E)$
$n \leftarrow H(\mathsf{ctx}), z_0 \leftarrow z_{\mathsf{U}0} + z_{\mathsf{I}0}, z_1 \leftarrow z_{\mathsf{U}1} + z_{\mathsf{I}1}$
$S \leftarrow F^{\mathsf{DY}}_{z_0}(\mathsf{ctr}),\ B \leftarrow F^{\mathsf{DY}}_{z_1}(\mathsf{ctr})$
$D \leftarrow g^{\mathsf{sk_U}} \cdot B^n$
$\alpha \xleftarrow{\$} \mathbb{Z}_p$
$(\sigma', m') \leftarrow \mathsf{SEQ.ChangeRep}(\sigma, m, \alpha)$
$(\tilde{g_1}, C'_\mathsf{F}, R', E') \leftarrow m'$
$\pi_\mathsf{T} \leftarrow \mathsf{PoK}\{\mathsf{sk_U}, \mathsf{ctr}, z_0, z_1, \mathsf{open}_{C_\mathsf{A}} :$
  $S = F^{\mathsf{DY}}_{z_0}(\mathsf{ctr}) \wedge D = g^{\mathsf{sk_U}} \cdot (F^{\mathsf{DY}}_{z_1}(\mathsf{ctr}))^n \wedge$
  $\mathsf{pk'_U} = g^{\mathsf{sk_U} \cdot \alpha} \wedge \tilde{g_1} = g_1^{\alpha} \wedge$
  $C'_\mathsf{F} = [(uh^r \prod_{i=1}^{\ell} y_i^{a_i})^{\alpha}; \mathsf{open}_{C_\mathsf{A}}] \wedge$
  $0 \leqslant \mathsf{ctr} < N \wedge \mathsf{P(A)} = 1\}$
return $T \leftarrow (\sigma', m', S, D, \pi_\mathsf{T})$

Algorithm $\mathsf{VerifyToken}(\mathsf{pp}, \mathsf{pk_I}, T, \mathsf{ctx}, \mathsf{P})$:

$(\sigma', m', S, D, \pi_\mathsf{T}) \leftarrow T$
$(\tilde{g_1}, C'_\mathsf{F}, R', E') \leftarrow m'$
if $\neg\mathsf{SEQ.Verify}(\mathsf{pk_I}, m', \sigma')$
  return 0
if $\neg\mathsf{VerifyPoK}(\pi_\mathsf{T}, S, D, N, \mathsf{pk'_U}, C'_\mathsf{F}, \mathsf{P})$
  return 0
return 1

Algorithm $\mathsf{VerifyCred}(\mathsf{pp}, \mathsf{pk_I}, \mathsf{pk_V}, \mathsf{pk_U}, \mathsf{cred}, \mathsf{req})$

if $\mathsf{cred} = \bot$:
  return $\bot$
$(\mathsf{credential}, C_\mathsf{A}, C_\mathsf{F}, z_{\mathsf{I}0}, z_{\mathsf{I}1}, \pi_\mathsf{E}) \leftarrow \mathsf{cred}$
$(\sigma, E) \leftarrow \mathsf{credential}$
$R \leftarrow H(\mathsf{req})$
if $\neg\mathsf{SEQ.Verify}(\mathsf{pk_I}, (g_1, C_\mathsf{F}, R, E), \sigma)$
  return $\bot$
if $\neg\mathsf{VerifyPoK}(\pi_\mathsf{E}, \mathsf{pk_V})$:
  return $\bot$
if $C_\mathsf{F} \neq C_\mathsf{A} \cdot y_{\ell-2}^{z_{\mathsf{I}0}} \cdot y_{\ell-1}^{z_{\mathsf{I}1}}$:
  return $\bot$
return $(R, z_{\mathsf{I}0}, z_{\mathsf{I}1}, \mathsf{credential})$

Algorithm $\mathsf{RecoverMetadata}(\mathsf{pp}, \mathsf{sk_V}, T, \mathsf{ctx})$:

$(x_0, x_1) \leftarrow \mathsf{sk_V}$
$(\sigma', m', S, D, \pi_\mathsf{T}) \leftarrow T$
$(\tilde{g_1}, C'_\mathsf{F}, R', E') \leftarrow m'$
for $b \in \{0, 1\}$ do
  $\tilde{E} \leftarrow R'^{(1-b)x_0 + bx_1}$
  if $\tilde{E} = E'$
    return b
end for
return $\bot$

Algorithm $\mathsf{IdentifyUser}(\mathsf{pp}, T_1, \mathsf{ctx}_1, T_2, \mathsf{ctx}_2)$:

Parse T_1 as $(\sigma_1, m_1, S_1, D_1, \pi_1)$
Parse T_2 as $(\sigma_2, m_2, S_2, D_2, \pi_2)$
if $S_1 \neq S_2 \vee \mathsf{ctx}_1 = \mathsf{ctx}_2$
  return $\bot$
$n_1 \leftarrow H(\mathsf{ctx}_1), \quad n_2 \leftarrow H(\mathsf{ctx}_2)$
$W \leftarrow (\frac{D_1}{D_2})^{\frac{1}{n_1 - n_2}} \pmod{\mathrm{q}}$
$\mathsf{pk_U} \leftarrow \frac{D_1}{W^{n_1}} \pmod{\mathrm{q}}$
return $\mathsf{pk_U}$

Fig. 8. Token generation and verification algorithms for the Cheshire construction.

5 Security Proofs

We present our main theorems for $\mathsf{SMA}^2\mathsf{RT}$ security of the Cheshire construction from Section 4. For each theorem we start by providing a proof sketch, and then go into the details of the full proof. We begin by noting that correctness follows by inspection.

Theorem 5.1 (Unforgeability). *If the SEQ signature scheme (Section 3) is secure, then the Cheshire construction satisfies the* $\mathsf{SMA}^2\mathsf{RT}$ *unforgeability definition (Definition 2.2).*

Proof (Sketch). We define a knowledge extractor $\mathcal{E}$ that uses standard Fiat-Shamir rewinding to extract the witness from the non-interactive proofs π_{A} and π_{T}. Specifically, for each valid request, $\mathcal{E}$ extracts the PRF seed z_0 and pre-computes the set of N valid serial numbers $\mathcal{Q}_{\mathsf{issuedCred}}$. Note that the extractor is given all of the issuer's secret keys which means that it will not encounter the nested rewinding issue, making concurrent security possible.

The proof proceeds by a reduction to the security of the underlying primitives for each of the five winning conditions:

Credential Forgery (Win 1) : The adversary outputs a token with a valid serial number not output by the extractor (i.e., not linked to a valid issued credential). We analyze this via three sub-cases based on the extracted witness:

- *Case 1A (Binding Violation)*: The token T has a fresh serial number $\mathsf{SN}(T)$ and contains a signed message m' that is equivalent to a message m that was signed by the challenger; $\mathsf{SN}(T)$ differs from those derived by $\mathcal{E}$ from the committed seed. We give a reduction that breaks the binding property of the blindable Pedersen commitment (and thus the DL assumption, which breaks the security of the SEQ scheme).
- *Case 1B (Signature Forgery)*: The token T has a fresh serial number $\mathsf{SN}(T)$ and contains a message m' not equivalent to any message signed by the challenger. We give a reduction that breaks unforgeability of the SEQ signature scheme. The interesting thing about this reduction is that, since it does not have the signing key (as indeed it is trying to break the security of the signature scheme), it cannot take advantage of black-box access to $\mathcal{A}$ to rewind it: any query from $\mathcal{A}$ requiring a signature may be problematic for this reduction. As a result, it cannot use the extractor $\mathcal{E}$. Yet, that handicap is immaterial, because the reduction does not in fact need to be able to extract anything: just the fact that $\mathcal{A}$ succeeds in producing an m' belonging to a heretofore unsigned equivalence class is sufficient to break unforgeability of the SEQ scheme.
- *Case 1C (Honest Key Recovery)*: The token reuses an honest user's serial number and contains a signed message $m' = (g_1', C_F', R', E')$ that is equivalent to a message m that was signed by the challenger when issuing a wallet for this honest user. We construct a reduction that, by extracting the opening to the blinded Pedersen commitment C_F' solves the discrete logarithm problem, and contradicts the security of the SEQ scheme.

Bounded Issuance Violation (Win 2): The adversary outputs more than N tokens for a single credential. Since the extractor $\mathcal{E}$ outputs exactly the set of N valid serial numbers for every issued credential, and we have already ruled out Win 1, this case is impossible.

Traceability Failure (Win 3): The adversary outputs valid tokens that frame a user created by the adversary, but the $\mathsf{IdentifyUser}$ algorithm fails to recover the correct public key. This implies the extracted seeds and secret key in the witness differ from those committed in the credential, which gives us a reduction that breaks the binding property of the Pedersen commitment.

Attribute Violation (Win 4): The adversary produces a valid token for a policy P that the committed attributes do not satisfy. Since the zero-knowledge proof verifies, P is satisfied, so the extracted attributes must differ from the committed ones. This also yields a reduction that breaks the binding property of the Pedersen commitment.

Metadata Manipulation (Win 5): The adversary outputs a token where the recovered metadata bit b^* differs from the embedded bit b. This implies the ciphertext in the token is not equivalent to the one signed by the issuer. This constitutes a forgery of the SEQ signature on a new equivalence class. □

Proof (Full Proof of Theorem 5.1). To provide a comprehensive analysis, we now detail the formal reductions for the five winning conditions.

First, we define the extractor $\mathcal{E}$ in more detail as follows. $\mathcal{E}$ receives as input the issuer's and the authorized verifier's secret keys, as well as the description of the adversary $\mathcal{A}$ and its fixed random tape. $\mathcal{E}$'s task is to extract the potential serial numbers of all the tokens that the adversary is able to produce. To do so, $\mathcal{E}$ runs the standard Fiat-Shamir extraction for the proof π_{A}: it resets $\mathcal{A}$ and issues a different Σ-protocol random-oracle challenge for its computation of π_{A}, and runs π_{A} for 2τ steps, where τ is the number of steps that $\mathcal{A}$ ran for in between when it issues a random oracle call that was used in π_{A} and when it produced π_{A}. When running $\mathcal{A}$ it needs the secret keys $\mathsf{sk_I}$ and $\mathsf{sk_V}$ in order to respond to $\mathcal{A}$'s queries to the oracles $\mathcal{O}_{\mathsf{Resp}}$ and $\mathcal{O}_{\mathsf{RecMet}}$. If in that time, $\mathcal{A}$ does not produce a proof π'_{A} from which the witness can be extracted, $\mathcal{E}$ tries again: it resets $\mathcal{A}$ and runs it for 2τ steps again until π'_{A} is obtained. Once the witness is extracted, the challenger obtains the user's seed z_{U_0}. Since the challenger itself samples z_{I_0} as part of running $\mathsf{Response}$, it can compute the final seed $z_0 = z_{U_0} + z_{I_0}$ and derive all N possible serial numbers $\{F^{\mathsf{DY}}_{z_0}(i)\}_{i=1}^{N}$. These serial numbers are stored in $\mathcal{Q}_{\mathsf{issuedCred}}$.

Now that $\mathcal{E}$ is defined, let $\mathcal{A}$ be a probabilistic poly-time adversary that wins the unforgeability experiment for the Cheshire scheme with extractor $\mathcal{E}$ with probability ε; without loss of generality, assume that $\mathcal{A}$'s random tape is fixed so that its success probability is ε over the random choices of the unforgeability experiment.

Recall the five distinct winning conditions defined in $\mathsf{Exp}^{\mathsf{unforge}}_{\mathsf{SMA^2RT}}$. As written in the game, the adversary outputs k forgeries of tokens, and each of these tokens is checked against the winning conditions.

We first direct our attention to **Win 1**, where the token's serial number does not correspond to any serial number output by the extractor; i.e. it is not linked to a credential issuing protocol. This occurs when $\mathcal{A}$'s output includes at least one valid tuple $(T_i^*, \mathsf{P}_i^*, \mathsf{ctx}_i^*)$ such that the serial number of this token $\mathsf{SN}(T_i^*)$ does not belong to the set of serial numbers associated with any issued credential in $\mathcal{Q}_{\mathsf{issuedCred}}$.

Recall that a token consists of a message $m' = (\tilde{g}_1, C'_{\mathsf{F}}, R', E')$, the randomized SEQ signature σ' on m', as well as a proof π_{T}. Let us consider a modified experiment $\mathsf{Exp}^{\mathsf{unforge-1A}}_{\mathsf{SMA^2RT}}$ in which, after $\mathcal{A}$ outputs a set of tokens that allow it to get to Win 1, the experiment rewinds the adversary to the step where it issued the random-oracle call that led it to eventually output π_{T}; the experiment produces a different response from the random oracle, and runs $\mathcal{A}$ to completion; if in this run $\mathcal{A}$ did not produce a proof π'_{T} that allows for the Σ-protocol's extraction of the opening r, α and $\mathsf{A} = (a_1, \ldots, a_\ell)$ of the commitment $C'_{\mathsf{F}} = (uh^r \prod_{i=1}^{\ell} y_i^{a_i})^{\alpha}$, as well as the value β such that $\tilde{g}_1 = g_1^{\beta}$ then the modified experiment $\mathsf{Exp}^{\mathsf{unforge-1A}}_{\mathsf{SMA^2RT}}$ rewinds the adversary and tries again. Eventually, with overwhelming probability, the modified experiment will obtain the opening r, α and $\mathsf{A} = (a_1, \ldots, a_\ell)$ to C'_{F}, as well as β and ctr such that $\mathsf{SN}(T_i^*) = F_{a_{\ell-2}}(\mathsf{ctr})$.

The witness will correspond to one of three cases:

Case 1A: Binding Violation by Reuse of Commitment. Suppose m' is equivalent to some $m_i = (g_1, C_{\mathsf{F}}, R, E)$ that was signed by the I in the query to the response oracle $\mathcal{O}_{\mathsf{Resp}}$. Suppose this case happens with probability ε_{1A}.

If ε_{1A} is non-negligible, then the binding property of the blindable Pedersen commitment scheme does not hold. The reduction $\mathcal{B}$ will take as input the parameters of the blindable Pedersen commitment scheme, and will generate all the other variables the same way as the challenger in $\mathsf{Exp}^{\mathsf{unforge}}_{\mathsf{SMA^2RT}}$.

The reduction $\mathcal{B}$ runs the modified experiment $\mathsf{Exp}^{\mathsf{unforge-1A}}_{\mathsf{SMA^2RT}}$ described above. Recall that $\mathsf{Exp}^{\mathsf{unforge-1A}}_{\mathsf{SMA^2RT}}$ extracts the opening r, α and $\mathsf{A} = (a_1, \ldots, a_\ell)$ of the commitment $C'_{\mathsf{F}} = (uh^r \prod_{i=1}^{\ell} y_i^{a_i})^{\alpha}$. It also extracts the value β such that $\tilde{g}_1 = g_1^{\beta}$. Since $m' \equiv m$, it must be the case that $C'_{\mathsf{F}} = (C_{\mathsf{F}})^{\beta}$. Thus, if $\mathsf{SN}(T_i^*) = F^{\mathsf{DY}}_{a'_{\ell-2}}(\mathsf{ctr}) \neq F^{\mathsf{DY}}_{a_{\ell-2}}(\mathsf{ctr})$ where $a_{\ell-2}$ is the seed z_0 that's part of the opening of the commitment in C_{F} (we know that $\mathsf{SN}(T_i^*)$ cannot be derived from any seed that corresponds to the opening of C_{F}). Thus, we have broken the binding property of the blindable Pedersen commitment (Section 3.2) which, in turn, violates the discrete logarithm assumption (Lemma 3.4).

Case 1B: Forgery of SEQ *Signature.* Consider the event that Win 1A occurs and yet the equivalence class represented by m' was *never* signed by the $\mathcal{O}_{\mathsf{Resp}}$ oracle or as part of creating a wallet for an honest user by the oracle $\mathcal{O}_{\mathsf{User}}$; let ε_{1B} be the probability of this event.

Let us construct a reduction $\mathcal{B}$ that breaks the unforgeability of the SEQ signature scheme whenever ε_{1B} is non-negligible.

Proof (Proof of Case 1B). First, it is important to note that our reduction $\mathcal{B}$ will play a slightly altered game with the adversary. The reduction does not

have the signing keys, and therefore cannot reset the adversary and rewind; so it cannot use the extractor $\mathcal{E}$. Instead, the reduction has access to the SEQ signing oracle $\mathcal{O}_{\mathsf{SEQ}}$ (as in the SEQ unforgeability experiment). $\mathcal{B}$ receives as input the SEQ public key $\mathsf{pk}_{\mathsf{SEQ}}$ from its challenger and sets $\mathsf{pk}_{\mathsf{I}} = \mathsf{pk}_{\mathsf{SEQ}}$. Then, $\mathcal{B}$ sets up the system parameters and oracles for $\mathcal{A}$ as follows:

- $\mathcal{A}$ handles $\mathcal{O}_{\mathsf{UKGen}}$ and $\mathcal{O}_{\mathsf{User}}$ exactly as described in the unforgeability experiment in Figure 2; it can do so since it knows the user keys.
- When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Resp}}$ with a req, then $\mathcal{B}$ runs all the instructions of $\mathcal{O}_{\mathsf{Resp}}$ except for the call to $\mathcal{E}$, as follows: as part of running the Response algorithm, it verifies the request sent by $\mathcal{A}$, encrypts the metadata bit, includes a PoK π_{E}, adds its own randomness to the commitment C_{A} to form C_{F}, and finally forms a message $m \leftarrow (g_1, C_{\mathsf{F}}, R, E)$. Then $\mathcal{B}$ forwards the message m to its own SEQ signing oracle to obtain an SEQ signature σ. It sets the credential as $\mathsf{credential} \leftarrow (\sigma, E)$ and finally returns the response $\mathsf{cred} \leftarrow (\mathsf{credential}, C_{\mathsf{A}}, C_{\mathsf{F}}, z_{\mathsf{I0}}, z_{\mathsf{I1}}, \pi_{\mathsf{E}})$ to $\mathcal{A}$.

Eventually, $\mathcal{A}$ will output k tokens $\{T_1^*, \cdots, T_k^*\}$ as its forgery. Now, if $\mathcal{A}$ wins via *Case B*, at least one of these tokens T_i^* contains a signature σ_i^* on a message m_i^* that was not queried to the signing oracle. Unlike *Case A*, we do not need to rewind here since we are assuming the standard unforgeability definition where a valid pair (m, σ) is sufficient for a forgery. However, since $\mathcal{A}$ outputs k tokens, then $\mathcal{B}$ must select the correct one to submit as its forgery. So, $\mathcal{B}$ picks a random index $j \in \{1, \cdots, k\}$ and outputs the message-signature pair (m_i^*, σ_i^*) from T_i^*.

Since the choice of i is independent of the adversary's view, whenever $\mathcal{A}$ succeeds in *Case B*, with probability $1/k$, $\mathcal{B}$ selects the correct forgery index. Putting it together, $\mathcal{B}$ breaks the unforgeability property of the SEQ signature scheme with probability ε_{1B}/k. [*End of proof of Case 1B.*]

Finally, there is one more case in which the serial number of one of the tokens is not among those produced by the extractor $\mathcal{E}$ that does not fall within the first two cases:

Case 1C: Recovering an honest user's secret key. We prove this case via a reduction where $\mathcal{B}$ breaks the discrete logarithm (DL) assumption given an adversary $\mathcal{A}$ can output a token using an honest user's secret key, sk_{U} with probability ε_{1C}.

Proof (Proof of Case 1C).

The reduction $\mathcal{B}$ interacts with the DL challenger that first generates the public parameters $(\mathbb{G}, q, g)$, samples a random value $x \xleftarrow{\$} \mathbb{Z}_q$, and sets $h = g^x$. The challenger forwards $(\mathbb{G}, q, g, h)$ to the reduction $\mathcal{B}$. The reduction $\mathcal{B}$ must answer $\mathcal{A}$'s queries to the $\mathcal{O}_{\mathsf{UKGen}}, \mathcal{O}_{\mathsf{User}}, \mathcal{O}_{\mathsf{Resp}}$, and $\mathcal{O}_{\mathsf{GenToken}}$. It does so honestly, except that in $\mathcal{O}_{\mathsf{UKGen}}$ for each user's key generation, it samples a random value $r_i \xleftarrow{\$} \mathbb{Z}_p$ and sets the public key of the user as $\mathsf{pk}_{\mathsf{U}} = h^{r_i}$. The reduction also needs to simulate all proofs for the user including π_{A} in $\mathcal{O}_{\mathsf{User}}$ and π_{T} in $\mathcal{O}_{\mathsf{GenToken}}$ since it does not know any user's secret keys. Due to the perfect zero-knowledge

property of the NIZK system (in the random oracle model), the distribution of the real proofs are identical to the simulated ones.

Eventually, $\mathcal{A}$ will output a forged token T_i. $\mathcal{B}$ uses the extractor $\mathcal{E}$ to extract a valid witness w_i from π_{T_i}. The witness w_i must include the openings to the commitment vector, specifically the secret key of the user sk_U. The knowledge extraction property of the proof system implies that $\mathsf{pk}_\mathsf{U} = g^{\mathsf{sk}_\mathsf{U}}$. Recall, that $\mathcal{B}$ outputted every honest user's public keys as $\mathsf{pk}_\mathsf{U} = h^{r_i}$ for some randomly chosen r_i for each user. Since $\mathcal{B}$ was able to extract the secret key sk_U from the witness, it means that the secret key must be xr_i since $\mathsf{pk}_\mathsf{U} = g^{\mathsf{sk}_\mathsf{U}} = h^{r_i} = g^{xr_i}$. $\mathcal{B}$ solves for x by dividing xr_i by r_i, and returns x to its DL challenger.

Since $\mathcal{A}$ was able to output a forgery by using an honest user's key with non-negligible probability, this implies that $\mathcal{B}$ is able to break the discrete logarithm with at least $\mathcal{A}$'s probability of ε_{1C}. [*End of proof of Case 1C.*]

Now we have covered all cases of a credential forgery that occurs in **Win 1**'s condition.

Next, we assume that **Win 1** did not happen and look at **Win 2**, where the adversary outputs more than N tokens derived from a single credential. In this case, we can use the analysis from *Case 1A*, where the extractor $\mathcal{E}$ extracts the seed z_0 that is used for the DY PRF that generates all the serial numbers that will correspond to a call to $\mathcal{O}_{\mathsf{Resp}}$. Since the game in $\mathsf{Exp}^{\mathsf{unforge-1A}}_{\mathsf{SMA^2RT}}$ explicitly disqualifies duplicate tokens (duplicate serial numbers with the same context), and we have already ruled out **Win 1** (where a serial number falls outside the extracted set), it follows that the adversary cannot produce $k > |\mathcal{Q}_{\mathsf{issuedCred}}| \cdot N$ valid tokens with distinct serial numbers because that's how many distinct serial numbers the extractor outputs. Therefore, $\mathsf{adv}^{\text{Win 2}}_{\mathcal{A}} = 0$.

Now, we look at win case **Win 3**. Recall from $\mathsf{Exp}^{\mathsf{unforge}}_{\mathsf{SMA^2RT}}$ this case occurs when the adversary outputs two tokens $T_i = (\sigma'_i, m'_i, S_i, D_i, \pi_{\mathsf{T}_i})$ and $T_j = (\sigma'_j, m'_j, S_j, D_j, \pi_{\mathsf{T}_j})$, with the same serial numbers $S_i = S_j$, but different contexts $\mathsf{ctx}_i \neq \mathsf{ctx}_j$, such that the $\mathsf{IdentifyUser}$ algorithm fails to correctly output the public key, pk_U (which must belong to an honest user). Suppose that this happens with probability ε_3. If ε_3 is non-negligible, then the binding property of the blindable Pedersen commitment scheme does not hold. Recall the reduction $\mathcal{B}$ from $\mathsf{Exp}^{\mathsf{unforge-1A}}_{\mathsf{SMA^2RT}}$; let us adapt it for this case as well: it will extract the openings of the commitment from both of the tokens with the same serial number. We know that the $\mathsf{IdentifyUser}$ algorithm consists of solving a linear system based on the double spending factor D that uses the DY PRF as follows: $D = g^{\mathsf{sk}_\mathsf{U}} \cdot F^{\mathsf{DY}}_{z_1}(\mathsf{ctr})^{H(\mathsf{ctx})}$. The reduction $\mathcal{B}$ extracts the values corresponding to the opening of C_F from $\mathsf{A} = (a_1, \ldots, a_\ell)$, where the last three values include the PRF seeds z_0 ($\mathsf{A}[a_{\ell-2}]$) and z_1 ($\mathsf{A}[a_{\ell-1}]$), and the user's secret key sk_U ($\mathsf{A}[a_\ell]$).

Since the tokens passed the $\mathsf{VerifyToken}$ algorithm, this means the double spending tags D_i and D_j were also computed correctly with respect to the witness that is extracted. Specifically, the base B is derived as $B = F^{\mathsf{DY}}_{z'_1}(\mathsf{ctr})$, and the tags are formed as: $D_i = g^{\mathsf{sk}'_\mathsf{U}} \cdot B^{H(\mathsf{ctx}_i)}$ and $D_j = g^{\mathsf{sk}'_\mathsf{U}} \cdot B^{H(\mathsf{ctx}_j)}$. The $\mathsf{IdentifyUser}$ algorithm solves the linear system defined by these two equations to recover pk_U. Since $\mathsf{ctx}_i \neq \mathsf{ctx}_j$, the system is linearly independent and must yield

the unique solution $\mathsf{pk}'_{\mathsf{U}} = g^{\mathsf{sk}'_{\mathsf{U}}}$. However, since $\mathsf{IdentifyUser}$ fails (outputs $\perp$ or an incorrect key), it means two things: that the extracted PRF seed z_1, is not equal to the committed value z'_1 in C'_{F} or that the extracted secret key sk_{U} does not equal the committed secret key $\mathsf{sk}'_{\mathsf{U}}$. Thus the reduction $\mathcal{B}$ produces two different openings to the same blindable Pedersen commitment C_{F} with probability ε_3.

Next, we address **Win 4**, which is the attribute violation. Recall that in $\mathsf{Exp}^{\mathsf{unforge}}_{\mathsf{SMA^2RT}}$, the adversary $\mathcal{A}$ outputs a valid token for a policy P^* and attributes A recorded in $\mathcal{Q}_{\mathrm{issuedCred}}$, yet $\mathsf{P}^*(\mathsf{A}) = 0$. Assume this occurs with a probability of ε_4. The reduction $\mathcal{B}$ follows similarly as in $\mathsf{Exp}^{\mathsf{unforge-1A}}_{\mathsf{SMA^2RT}}$, where $\mathcal{B}$ runs the knowledge extractor $\mathcal{E}$ on the zero-knowledge proof π_{T} contained in the token to extract the witness A' that satisfies the policy P^*. We know that the extracted witness contains the opening of the commitment A', which must satisfy the policy verification equation: $\mathsf{P}^*(\mathsf{A}') = 1$ and that $C'_{\mathsf{F}} = C^{\alpha}_{\mathsf{F}}$. However, this is not the same opening as committed in the credential, because we know that $\mathsf{P}^*(\mathsf{A}) = 0$, which implies that $\mathsf{A} \neq \mathsf{A}'$. Again, the adversary was able to form two different openings to the same multi-base Pedersen commitment C_{F} with probability ε_4.

Finally, we move to the last winning case, **Win 5**, where the adversary outputs a valid token T^* such that the recovered metadata bit b^* differs from the embedded bit b. We construct the reduction that runs similarly to $\mathsf{Exp}^{\mathsf{unforge1B}}_{\mathsf{SMA^2RT}}$, such that $\mathcal{B}$ breaks the unforgeability of the signature scheme SEQ whenever ε_5 is non-negligible.

Again like in *Case 1B*, the reduction does not have the signing keys, and therefore cannot reset the adversary and rewind; so it cannot use the extractor $\mathcal{E}$. Instead, the reduction has access to the SEQ signing oracle $\mathcal{O}_{\mathsf{SEQ}}$ (as in the SEQ unforgeability experiment). $\mathcal{B}$ runs the same as in *Case 1B*. Since E is the value that represents an encryption of b, then any valid re-randomization of the ciphertext in the same equivalence class E' must decrypt to the same b. If $\mathsf{RecoverMetadata}$ outputs $b^* \neq b$, then E' decrypts to a different value, meaning E' is not equivalent to E. Consequently, the message m' (which contains E') is not equivalent to the originally signed message m. This implies that the adversary has produced a valid signature σ' on a new message class m' that was never queried to the signing oracle. This is a forgery against the SEQ signature scheme and thus the adversary wins with probability ε_5

Combining all these cases, we conclude the advantage of the adversary is negligible. □

Theorem 5.2 (Privacy of metadata bit). *If the Decisional Diffie-Hellman (DDH) assumption holds in the group $\mathbb{G}$, then the Cheshire construction satisfies the $\mathsf{SMA^2RT}$ Privacy of Metadata Bit definition (Definition 2.4).*

First, we clarify the role of the extractor $\mathcal{E}$ and the set $\mathcal{L}^*$ in the experiment. In the $\mathcal{O}_{\mathsf{RecMet}}$ oracle, the challenger must determine whether a given token originated from the challenge credential return $\perp$ for such tokens. To do so, the challenger needs the serial numbers that the challenge credential can produce. Recall that serial numbers are computed as $S = F^{\mathsf{DY}}_{z_0}(\mathsf{ctr})$ for $\mathsf{ctr} \in [0, N)$.

When the adversary $\mathcal{A}$ submits a challenge request req^* to the $\mathcal{O}_{\mathsf{Chal}}$ oracle, the challenger runs the knowledge extractor $\mathcal{E}$ on the zero-knowledge proof π_{A}

contained in req^* to recover the user's seed z_{U_0}. Specifically, $\mathcal{E}$ performs the standard Fiat-Shamir extraction: it resets $\mathcal{A}$ and issues a different Σ-protocol random-oracle challenge for its computation of π_{A}, and runs $\mathcal{A}$ for 2τ steps, where τ is the number of steps that $\mathcal{A}$ ran for in between when it issued the random oracle call used in π_{A} and when it produced π_{A}. When running $\mathcal{A}$ it needs the secret keys sk_{I} and sk_{V} in order to respond to $\mathcal{A}$'s queries to the oracles $\mathcal{O}_{\mathsf{Resp}}$ and $\mathcal{O}_{\mathsf{RecMet}}$. If in that time $\mathcal{A}$ does not produce a proof π'_{A} from which the witness can be extracted, $\mathcal{E}$ resets $\mathcal{A}$ and tries again until extraction succeeds. Once the witness is extracted, the challenger obtains the user's seed z_{U_0}. Since the challenger itself samples z_{I_0} as part of running $\mathsf{Response}$, it can compute the final seed $z_0 = z_{U_0} + z_{I_0}$ and derive all N possible serial numbers $\{F^{\mathsf{DY}}_{z_0}(i)\}_{i=0}^{N-1}$. These serial numbers are stored in $\mathcal{L}^*$. When queried, the $\mathcal{O}_{\mathsf{RecMet}}$ oracle then checks whether the serial number of any queried token belongs to $\mathcal{L}^*$; if so, the token originates from the challenge credential and the oracle returns $\perp$.

We prove this theorem using a hybrid argument. We define a sequence of games, starting from the real experiment with $\beta = 0$ and ending with $\beta = 1$, showing that each transition is computationally indistinguishable. We begin with a proof sketch and then provide the full proof.

Proof (sketch).

Game 0. This is the real experiment $\mathsf{Exp}^{\mathsf{privacyMB}}_{\mathsf{SMA^2RT}}$, where the challenger $\mathcal{C}$ always selects $\beta = 0$. The ciphertext is computed as $(R, E) = (H(\mathsf{req}), R^{x_0})$ and the proof π_{E} is generated honestly. The $\mathcal{O}_{\mathsf{RecMet}}$ oracle answers queries using the secret key $\mathsf{sk}_{\mathsf{V}} = (x_0, x_1)$.

↕ *Claim 1: identical in the random oracle model (algebra)*

Game 1. Identical to Game 0, except the challenger programs the random oracle, recording the discrete logarithm ρ such that $H(\mathsf{req}) = g_1^{\rho}$ for each issuance query. It stores (ρ, b) in a lookup table. The $\mathcal{O}_{\mathsf{RecMet}}$ oracle uses this table to match a token $T = (\sigma', m', S, D, \pi_{\mathsf{T}})$ to a secret metadata bit provided at the time the adversary called $\mathcal{O}_{\mathsf{Resp}}$: it parses $m' = (\tilde{g}_1, C'_F, R', E')$, and for each entry (ρ, b) in its table, check whether $E' = (R')^{\rho}$, and if so, return b. If no entry like that is found, the challenger falls back to using the $\mathsf{RecoverMetadata}$ algorithm on input sk_{V}.

↕ *Claim 2: SEQ unforgeability*

Game 2. Identical to Game 1, except the fallback to using $\mathsf{RecoverMetadata}$ in $\mathcal{O}_{\mathsf{RecMet}}$ is removed. The oracle now uses only the discrete logarithm table: tokens matched to a session return the recorded bit, challenge tokens return $\perp$, and unmatched tokens return $\perp$. By SEQ unforgeability, every valid token matches some entry, so the fallback can only be triggered if SEQ is forgeable.

↕ *Claim 3: zero-knowledge property*

Game 3. Identical to Game 2, but all zero-knowledge proofs π_{E} are replaced by simulated proofs using the NIZK simulator.

↕ *Claim 4: DDH assumption*

Game 4. Identical to Game 3, except the challenge ciphertext E^* is replaced with a random group element g^z for $z \in \mathbb{Z}_p$.

↕ *Claim 5: DDH assumption*

Game 5. Identical to Game 4, but encrypt the challenge ciphertext with $\beta = 1$, so we switch the random E^* back to R^{*x_1}.
↕ *Claim 6: zero-knowledge property (symmetric to Claim 3)*
Game 6. Identical to Game 5, but simulated proofs are replaced with real proofs.
↕ *Claim 7: SEQ unforgeability (symmetric to Claim 2)*
Game 7. Identical to Game 6, but $\mathcal{O}_{\mathsf{RecMet}}$ is restored to use the discrete logarithm table with RecoverMetadata as a fallback for unmatched tokens.
↕ *Claim 8: identical in the random oracle model (symmetric to Claim 1)*
Game 8. Identical to Game 7, but the $\mathcal{O}_{\mathsf{RecMet}}$ oracle is restored to use the verifier's secret key (x_0, x_1) via RecoverMetadata, instead of the lookup table of discrete logarithms. This is the real experiment for $\beta = 1$.

Since the adversary's view in Game 0 is computationally indistinguishable from their view in Game 8, the adversary's advantage in distinguishing $\beta = 0$ from $\beta = 1$ is negligible.

□

Proof. **Game 0.** This is the real experiment $\mathsf{Exp}^{\mathsf{privacyMB}}_{\mathsf{SMA^2RT}}$ where the challenger always selects $\beta = 0$. The ciphertext (R, E) is computed as $E = R^{x_0}$ where $R = H(\mathsf{req})$, and the proof π_E is generated honestly for all issuance queries (both challenge and non-challenge). The $\mathcal{O}_{\mathsf{RecMet}}$ oracle answers queries using the verifier's secret key $\mathsf{sk_V} = (x_0, x_1)$ via the RecoverMetadata algorithm directly.
Game 1. Identical to Game 0, except that the challenger programs the random oracle H. For each query to $\mathcal{O}_{\mathsf{Resp}}$ or $\mathcal{O}_{\mathsf{Chal}}$ involving a request req, the challenger samples $\rho \xleftarrow{\$} \mathbb{Z}_p$ and sets $H(\mathsf{req}) = g_1^\rho$. It records the entry (ρ, b) in a table $\mathcal{T}$, where b is the metadata bit used for that session. For the challenge query, the challenger records ρ^* separately.

The $\mathcal{O}_{\mathsf{RecMet}}$ oracle now operates as follows: given a token T with randomized message components $(\tilde{g}_1, C'_F, R', E')$, the oracle first checks whether $R' = \tilde{g}_1^\rho$ for some entry $(\rho, b) \in \mathcal{T}$. If a match is found, it returns the recorded bit b. If $R' = \tilde{g}_1^{\rho^*}$, the token originates from the challenge credential, and the oracle returns $\perp$. For any remaining valid token that matches no entry, the oracle falls back to using RecoverMetadata using the V's secret key (x_0, x_1).
Claim 1: *The adversary's view in Game 0 is identically distributed to its view in Game 1 (by algebra).*

Proof (Proof of Claim 1). In the random oracle model, the challenger is free to program H by choosing its outputs; sampling $\rho \xleftarrow{\$} \mathbb{Z}_p$ and setting $H(\mathsf{req}) = g_1^\rho$ produces a uniformly random group element, which is identically distributed to a random oracle output.

For tokens matched by the table $\mathcal{T}$, the recorded bit b is the same bit that was used to form the ciphertext $E = R^{(1-b)x_0 + bx_1}$ during issuance. Since randomization preserves the equivalence class, we have $E' = R'^{(1-b)x_0 + bx_1}$, and so RecoverMetadata using (x_0, x_1) would return the same b. Thus the table lookup gives the identical answer.

For tokens not matched by the table: we fallback to using RecoverMetadata which is the same strategy as used in Game 0. For tokens where E' is neither

R'^{x_0} nor R'^{x_1}, both games return $\perp$ after the $\mathsf{VerifyToken}$ check fails. Thus the two games are identically distributed.

Game 2. Identical to Game 1, except that the fallback to $\mathsf{RecoverMetadata}$ in the $\mathcal{O}_{\mathsf{RecMet}}$ oracle is removed. The oracle now only uses the discrete logarithm table $\mathcal{T}$ and the challenge value ρ^*: for a token with components $(\tilde{g}_1, C'_F, R', E')$, it checks $R' = \tilde{g}_1^{\rho}$ against entries in $\mathcal{T}$ and returns the corresponding b if matched, returns $\perp$ if $R' = \tilde{g}_1^{\rho^*}$, and returns $\perp$ if no match is found.
Claim 2: *Under the unforgeability of the SEQ signature scheme, a PPT adversary cannot distinguish Game 1 from Game 2, except with negligible probability.*

Proof (Proof of Claim 2). The only difference between Games 1 and 2 is the behavior of $\mathcal{O}_{\mathsf{RecMet}}$ on valid tokens that match no entry in $\mathcal{T}$ and do not match ρ^*. We argue that such tokens exist only with negligible probability.

Suppose a token T passes $\mathsf{VerifyToken}$. Then T contains a message $\mathbf{m}' = (\tilde{g}_1, C'_F, R', E')$ and a signature σ' such that $\mathsf{SEQ.Verify}(\mathsf{pk_I}, \mathbf{m}', \sigma') = 1$. By the unforgeability of the SEQ signature scheme, $\mathbf{m}'$ must lie in the equivalence class of some message $\mathbf{m} = (g_1, C_F, R, E)$ that was signed by the issuer during either an $\mathcal{O}_{\mathsf{Resp}}$ or $\mathcal{O}_{\mathsf{Chal}}$ query. This means that, $\mathbf{m}' = \mathbf{m}^{\alpha}$ for some $\alpha \in \mathbb{Z}_p$, which gives $\tilde{g}_1 = g_1^{\alpha}$ and $R' = R^{\alpha}$. Since $R = H(\mathsf{req}) = g_1^{\rho}$ for some ρ recorded in $\mathcal{T}$ (or ρ^* for the challenge), we have:

$$R' = R^{\alpha} = (g_1^{\rho})^{\alpha} = (g_1^{\alpha})^{\rho} = \tilde{g}_1^{\rho}$$

Therefore, $R' = \tilde{g}_1^{\rho}$ and the token is always matched by the table $\mathcal{T}$ or by ρ^*. The fallback to using $\mathsf{RecoverMetadata}$, is never needed for valid tokens.

If a valid token were to match no entry, it would imply that $\mathbf{m}'$ lies in an equivalence class that was never signed, which is a forgery against the SEQ scheme. Thus, the probability that Games 1 and 2 differ is bounded by $\mathsf{Adv}^{\mathsf{SEQ\text{-}UF}}(\lambda)$, which is negligible. [*End of proof of Claim 2.*]

Game 3. Identical to Game 2, except that *all* zero-knowledge proofs π_E (in both $\mathcal{O}_{\mathsf{Resp}}$ and $\mathcal{O}_{\mathsf{Chal}}$) are replaced by simulated proofs using the NIZK simulator.
Claim 3: *The adversary's view in Game 2 is identically distributed to its view in Game 3.*

Proof (Proof of Claim 3). The only difference between Games 2 and 3 is that the honest NIZK proofs π_E are replaced by simulated ones. Since these are Fiat-Shamir-based proofs of discrete-log relations in prime-order groups, the perfect zero-knowledge property in the random oracle model guarantees that the distribution of real proofs is identical to that of simulated proofs. The $\mathcal{O}_{\mathsf{RecMet}}$ oracle is unaffected, as it operates solely on the discrete logarithm table $\mathcal{T}$ and does not depend on π_E. The rest of the algorithm follows as in the $\mathsf{Response}$ algorithm in our protocol. This includes generating the issuer random seeds to add to the attribute commitment, forming the message, signing it, and finally outputting the response cred.

Game 4. Identical to Game 3, except that in the challenge oracle $\mathcal{O}_{\mathsf{Chal}}$, the ciphertext component E^* is replaced with a uniformly random group element g^z for $z \xleftarrow{\$} \mathbb{Z}_p$. The simulated proof π_E is constructed to attest that this random E^* is a valid encryption.

Claim 4: *Under the DDH assumption, a PPT adversary cannot distinguish Game 3 from Game 4, except with negligible probability.*

Proof (Proof of Claim 4). First, the DDH challenger generates the group parameters $\mathbb{G}$ of order p with generator g. It samples random values $x, y, z \in \mathbb{Z}_p$ and a random bit $b \in \{0, 1\}$. It computes $X = g^x$ and $Y = g^y$. Then, it computes the challenge term Z based on the bit b. If $b = 0$, it sets $Z = g^{x \cdot y}$ (the real DDH tuple). If $b = 1$, it sets $Z = g^z$ (the random tuple). The challenger sends the tuple (g, X, Y, Z) to the reduction $\mathcal{B}$.

$\mathcal{B}$ simulates the environment for $\mathcal{A}$ as follows: it sets $g_1 = Y$ as part of the setup.

- Key generation: $\mathcal{B}$ sets the issuer's public key component $X_0 \leftarrow X$. Note that $\mathcal{B}$ does not know the corresponding secret key x_0. $\mathcal{B}$ samples $x_1 \xleftarrow{\$} \mathbb{Z}_p$ itself, sets $\mathsf{pk}_{\mathsf{V}} = (X, g^{x_1})$, and generates all other system parameters (including sk_{I} and pk_{I}) honestly.
- Oracle Queries to $\mathcal{O}_{\mathsf{Resp}}$: When $\mathcal{A}$ makes a query $\mathcal{O}_{\mathsf{Resp}}(\mathsf{pk}_{\mathsf{U}}, \mathsf{req}, b, \mathsf{pol})$, $\mathcal{B}$ programs the random oracle by sampling $\rho \xleftarrow{\$} \mathbb{Z}_p$ and setting $H(\mathsf{req}) = g_1^{\rho}$, so that $R = g_1^{\rho}$. To compute the ciphertext without knowledge of x_0:
  - If $b = 0$: $\mathcal{B}$ computes $E = R^{x_0} = (g_1^{\rho})^{x_0} = (g^{x_0})^{\rho} = X^{\rho}$. Since $\mathcal{B}$ knows X and ρ, it can compute E.
  - If $b = 1$: $\mathcal{B}$ computes $E = R^{x_1}$ directly, since it knows x_1.

  Just as in both games, $\mathcal{B}$ (lacking the value x_0) runs the NIZK simulator to generate the proof π_{E}, ensuring the response is indistinguishable from a valid one (as in Games 3 and 4). $\mathcal{B}$ records (ρ, b) in $\mathcal{T}$.
- Oracle Query to $\mathcal{O}_{\mathsf{Challenge}}$: When $\mathcal{A}$ requests the challenge token for a request req^*, $\mathcal{B}$ embeds the DDH challenge values into the token. $\mathcal{B}$ first ensures that req^* has never been queried to H before. Then, $\mathcal{B}$ programs the random oracle H such that $H(\mathsf{req}^*) = Y$. $\mathcal{B}$ sets the ciphertext component $E^* = Z$. It runs the NIZK simulator to generate the proof π_{E} attesting that (R^*, E^*) is a valid encryption. Because the proof is simulated, $\mathcal{B}$ does not need the discrete log of E^* to generate π_{E}. $\mathcal{B}$ then forms the rest of the request req^* in the same way as in Game 3. Note that the formal game specifies that $\mathcal{O}_{\mathsf{Chal}}$ employs a knowledge extractor $\mathcal{E}$ to populate $\mathcal{L}^*$ with the serial numbers of the challenge credential for use by $\mathcal{O}_{\mathsf{RecMet}}$. However, since we are in Game 3 where $\mathcal{O}_{\mathsf{RecMet}}$ operates using only the discrete logarithm table $\mathcal{T}$ (as established by the transition from Game 1 to Game 2), the reduction does not require the extractor's output. Challenge tokens are instead identified by elimination from $\mathcal{T}$, as described below.
- Oracle Query to $\mathcal{O}_{\mathsf{RecMet}}$: When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{RecMet}}(T, \mathsf{ctx})$, $\mathcal{B}$ first verifies the token via $\mathsf{VerifyToken}$. Then, $\mathcal{B}$ parses the token to obtain the randomized message $(\tilde{g}_1, C'_F, R', E')$ and checks whether $R' = \tilde{g}_1^{\rho}$ for each $(\rho, b) \in \mathcal{T}$. If a

match is found, $\mathcal{B}$ returns the corresponding bit b. If no match is found and the token is valid, then the token must originate from the challenge credential (since $\mathcal{B}$ programmed the random oracle for all queries, every valid R' must correspond to some issuance session by the argument in Claim 2), and $\mathcal{B}$ returns $\perp$. Note that this correctly simulates $\mathcal{O}_{\mathsf{RecMet}}$ without knowledge of sk_{V}, because challenge tokens are detected by elimination from the table $\mathcal{T}$.

Finally, $\mathcal{A}$ will output a bit b'. $\mathcal{B}$ will output the same bit to its DDH challenger.

If the DDH challenger chose $b = 0$, then $Z = g^{x \cdot y}$. Recall that $g_1 = Y = g^y$, so $R^* = g_1 = g^y$ and $E^* = Z = g^{xy} = R^{*x_0}$. Thus the challenge ciphertext is a valid encryption under x_0, and the resulting token is distributed identically to the challenge token in Game 3 (where the metadata bit $\beta = 0$).

If, instead, $b = 1$, then $Z = g^z$ for a random z, so $E^* = Z = g^z$. Since z is uniformly random and independent of all other values, z is also uniformly random, and thus E^* is a uniformly random group element. This corresponds exactly to how the challenger forms the challenge token in Game 4, where E^* is random. Therefore, if $\mathcal{A}$ can distinguish Game 3 from Game 4 with non-negligible advantage, $\mathcal{B}$ can solve the DDH problem with the same advantage. [*End of proof of Claim 4.*]

Game 5. In this game, the random ciphertext is replaced by an encryption under the second key X_1. Game 5 is the same as Game 4, except that the challenger now computes $E^* = R^{*x_1}$ (encrypting for $\beta = 1$). The proof π_{E} is still simulated.
Claim 5: *Under the DDH assumption, a PPT adversary cannot distinguish Game 4 from Game 5, except with negligible probability.*

Here, the reduction algorithm $\mathcal{B}'$ solves the DDH problem given a distinguisher between Games 4 and 5. The argument is symmetric to Claim 4, except that here, $\mathcal{B}'$ does not know the secret key x_1, only x_0.
Game 6. In this game, the simulated proofs are replaced by real proofs. Game 6 is the same as Game 5, except that the challenger generates all proofs π_{E} honestly using the appropriate V's secret key (either x_0 or x_1 depending on the metadata bit). For the challenge oracle, the proof is generated honestly using the secret key x_1. The $\mathcal{O}_{\mathsf{RecMet}}$ oracle continues to use only the discrete logarithm table $\mathcal{T}$.
Claim 6: *A PPT adversary cannot distinguish Game 5 from Game 6, except with negligible probability.*

The argument is symmetric to Claim 3. The only difference is whether π_{E} is simulated or generated honestly. By the perfect zero-knowledge property in the random oracle model, these distributions are identical. The $\mathcal{O}_{\mathsf{RecMet}}$ oracle is unaffected, as it does not depend on π_{E}.

Game 7. Identical to Game 6, except that the $\mathcal{O}_{\mathsf{RecMet}}$ oracle is restored to use the discrete logarithm table $\mathcal{T}$ as its primary lookup, with a fallback to $\mathsf{RecoverMetadata}$, for any unmatched token. That is, for tokens not matched by $\mathcal{T}$ or ρ^*, the oracle checks whether $E' = R'^{x_1}$; if so, it returns 1; otherwise it returns 0.

Claim 7: *Under the unforgeability of the SEQ signature scheme, a PPT adversary cannot distinguish Game 6 from Game 7, except with negligible probability.*

The argument is symmetric to Claim 2. By SEQ unforgeability, every valid token matches some entry in $\mathcal{T}$ or corresponds to ρ^*, so the fallback to $\mathsf{RecoverMetadata}$ is never triggered. Adding it back does not change the adversary's view.

Game 8. Identical to Game 7, except that the $\mathcal{O}_{\mathsf{RecMet}}$ oracle is restored to use the V's secret key $\mathsf{sk}_{\mathsf{V}} = (x_0, x_1)$ via the $\mathsf{RecoverMetadata}$ algorithm, as in the real experiment. The challenger no longer needs to maintain the discrete logarithm table $\mathcal{T}$. This is exactly the real experiment $\mathsf{Exp}^{\mathsf{privacyMB}}_{\mathsf{SMA^2RT}}$ with $\beta = 1$.
Claim 8: *The adversary's view in Game 7 is identically distributed to its view in Game 8.*

The argument is symmetric to Claim 1.

Since the adversary's view in Game 0 (where $\beta = 0$) is computationally indistinguishable from their view in Game 8 (where $\beta = 1$) through this sequence of games, the adversary's advantage in guessing the correct bit (either 0 or 1) is $\frac{1}{2} + \varepsilon$, which is negligible under the DDH assumption and the unforgeability of the SEQ signature scheme. □

Theorem 5.3 (Unlinkability). *If the Decisional Diffie-Hellman (DDH) assumption holds in the group $\mathbb{G}$, and the DY construction is a PRF, then the Cheshire construction satisfies the $\mathsf{SMA^2RT}$ unlinkability (Definition 2.5).*

Proof (Sketch). To prove unlinkability, we show that an adversary $\mathcal{A}$ cannot distinguish between the real experiment ($\beta = 0$) and a simulated experiment ($\beta = 1$). We define a simulator $\mathcal{S}$ that uses extracted secret keys $\mathsf{sk}_{\mathsf{I}}, \mathsf{sk}_{\mathsf{V}}$ (the trapdoor TD) to produce tokens that are independent of a user's identity or attributes. The proof proceeds through a sequence of games:
Game 0. Real world ($\beta = 0$). All oracles produce outputs as in protocol.
↕ *Claim 1: zero-knowledge property*
Game 1. All proofs are simulated.
↕ *Claim 2: security of the PRF*
Game 2. All PRF's are replaced by truly random functions.
↕ *Claim 3: identical by inspection*
Game 3. In $\mathcal{O}_{\mathsf{GenToken}}$, S and D are outputs of a truly random function.
↕ *Claim 4: class hiding and perfect adaptation property of SEQ signatures. This is the most involved step, more of that below.*
Game 4. In $\mathcal{O}_{\mathsf{GenToken}}$, ensure that the credential has not exceeded its limit on the number of tokens it can produce. Then instead of calling $\mathsf{GenToken}$, call $\mathcal{S}_{\mathsf{GenToken}}$ as defined above. Now the token generation is completely simulated.
↕ *Claim 5: hiding property of Pedersen commitment*
Game 5. All $\mathcal{O}_{\mathsf{Req}}$ outputs are simulated as described in $\mathcal{S}_{\mathsf{Request}}$. This is the simulated world ($\beta = 1$)

Since Game 0 and Game 5 are computationally indistinguishable, the advantage of any PPT adversary in the unlinkability experiment is negligible. □

Proof (Full Proof of Theorem 5.3). We first define a simulator $\mathcal{S}$ that interacts with the adversary $\mathcal{A}$ in the simulated world (where $\beta = 1$). $\mathcal{S}$ samples the group elements for the public parameters and sends them to $\mathcal{A}_1$. Once $\mathcal{A}_1$ runs the issuer and verifier key generation algorithms, it outputs the public keys $\mathsf{pk_I}$ and $\mathsf{pk_V}$ along with zero knowledge proofs of correct computation. Then, $\mathcal{S}$ uses oracle access to $\mathcal{A}_1$ to extract the secret keys $\mathsf{sk_V}$ and $\mathsf{sk_I}$ from these proofs, producing the trapdoor, TD, that it will need to form tokens. The simulator runs the functions $\mathcal{S}_{\mathsf{Setup}}, \mathcal{S}_{\mathsf{KeyGen_U}}, \mathcal{S}_{\mathsf{Request}}, \mathcal{S}_{\mathsf{GenToken}}$ in $\mathsf{Exp}^{\mathsf{unlink}}_{\mathsf{SMA^2RT},\mathcal{A},\mathcal{S}}(\lambda)$ as follows:

- $(\mathsf{pp}, \mathsf{state}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{Setup}}(1^\lambda, N)$: The simulator sets up the public parameters and stores its state as $\mathsf{state}_{\mathcal{S}}$.
- $(\mathsf{pk_U}, \perp) \leftarrow \mathcal{S}_{\mathsf{KeyGen_U}}(\mathsf{pp}_1)$: The simulator runs this algorithm honestly as in the algorithm $\mathsf{KeyGen_U}$ defined in our protocol, except that it does not store the secret key of the user, and outputs $\perp$ for the user's state.
- $\mathsf{req} \leftarrow \mathcal{S}_{\mathsf{Request}}(\mathsf{pp}, \mathsf{pk_U}, \mathsf{pol})$: To form a request, the simulator needs to simulate a commitment to the attributes A. To do so, it samples C_{A} at random from the group, and then runs the zero-knowledge simulator to generate a proof π_{A} of correct commitment to the attributes. Then the simulator computes $\mathsf{req} \leftarrow (\mathsf{pk_U}, C_{\mathsf{A}}, \pi_{\mathsf{A}})$. It also stores this information.
- $T \leftarrow \mathcal{S}_{\mathsf{GenToken}}(b, \mathsf{ctx}, \mathsf{P}, \mathsf{TD})$: For every token generation query, $\mathcal{S}$ must return a simulated token. It is important that this token be computed in a way that is independent of the user's identity or attributes, only on the hidden metadata bit. First, $\mathcal{S}$ samples the serial number S and the double-spending tag D as truly random elements from the target group $\mathbb{G}_1$. Next, it samples the message $m = (g_1', C_{\mathsf{A}}, R, E)$ as follows: g_1', C_{A} and R are uniformly random group elements, while $E = R^{x_b}$; then it uses the signing key $\mathsf{sk_I}$ to compute the signature σ on m. Finally, the simulator computes the simulated proof π_{T}, and outputs the resulting token $T = (\sigma, m, S, D, \pi_{\mathsf{T}})$.

We construct a sequence of games beginning with the real unlinkability experiment (Game 0), where $\beta = 0$ in $\mathsf{Exp}^{\mathsf{unlink}}_{\mathsf{SMA^2RT},\mathcal{A},\mathcal{S}}(\lambda)$, and described in Figure 3. In the real world, the oracles respond exactly as in the Cheshire construction: the public parameters in Setup are honestly generated, queries to $\mathcal{O}_{\mathsf{UKGen}}$ are honestly formed using $\mathsf{KeyGen_U}$, responses are honestly formed in $\mathcal{O}_{\mathsf{Req}}$ as in $\mathsf{Response}$, and the tokens are formed in $\mathcal{O}_{\mathsf{GenToken}}$ as in $\mathsf{GenToken}$. The sequence of games ends with a completely simulated game, where $\beta = 1$ in which all algorithms $\mathcal{S}_{\mathsf{Setup}}, \mathcal{S}_{\mathsf{KeyGen_U}}, \mathcal{S}_{\mathsf{Request}}, \mathcal{S}_{\mathsf{GenToken}}$, are simulated using truly random functions instead of the DY PRF, randomly sampled group elements in place of commitments to user attributes, and simulated proofs (Game 5). Our goal is to show that Game 0 and Game 5 are indistinguishable via a sequence of intermediate games.

Game 0. This is the real experiment $\mathsf{Exp}^{\mathsf{unlink}}_{\mathsf{SMA^2RT}}$, where $\beta = 0$. Here there is no simulation and all oracles respond with the algorithms described in the protocol.

Game 1. In this game, all proofs are simulated.

Game 1 is the same as Game 0, except that the challenger $\mathcal{C}$ simulates the ZK-PoK π_{A} and π_{T}.

Claim 1: *A PPT adversary cannot distinguish Game 0 from Game 1, except with negligible probability.*

The only difference between the two games is zero-knowledge simulation. In Game 0, the challenger generates real proofs in $\mathcal{O}_{\mathsf{Req}}$ where it outputs π_{A} to prove the attributes were honestly committed in C_{A}. In $\mathcal{O}_{\mathsf{GenToken}}$ the challenger produces a proof π_{T} to prove correct computation of the serial number, double spending equation, along with correctly committed values, predicate satisfaction, and proves the toke counter is within range. In contrast, the simulator will simulate the proofs π_{A} and π_{T} in Game 1. Since these are Fiat-Shamir-based proofs of discrete-log relations in prime-order groups in the random-oracle model, the distribution of the real proofs in Game 0 is identical to the simulated proofs in Game 1.

Game 2. In this game, all instances of the PRF evaluation are replaced by a truly random function.

Specifically, when the oracle $\mathcal{O}_{\mathsf{GenToken}}$ needs to evaluate the serial number $S = F^{\mathsf{DY}}_{z_0}(\mathsf{ctr})$ and the blinding factor $B = F^{\mathsf{DY}}_{z_1}(\mathsf{ctr})$, it will replace them with a truly random function, G and P from the PRF range, such that $S = G(\mathsf{ctr})$ and $B = P(\mathsf{ctr})$. $\mathcal{C}$ will compute these random functions as follows: it maintains a lookup table of inputs and outputs to both random functions. If an input has been queried before, it retrieves the previous output from the table; otherwise, it samples a new random value from the DY PRF range and adds this new input-output pair to the table.

Claim 2. *A PPT adversary cannot distinguish Game 1 from Game 2, except with negligible probability.*

The indistinguishability of Game 1 and Game 2 follows directly from the pseudorandomness property of the DY PRF. We construct a reduction $\mathcal{B}$ that breaks the PRF security given a distinguisher $\mathcal{A}$ for the games.

Proof (Proof of claim 2). First, note that, by a standard hybrid argument, it is sufficient to show a reduction that distinguishes an oracle that contains a collection of random functions (two functions, G and P for each wallet) from an oracle that contains a collection of DY PRFs: two per wallet.

The reduction $\mathcal{B}$ interacts with a the challenger that provides access to the oracle $\mathcal{O}$, which is either a collection of DY PRF functions $F^{\mathsf{DY}}_k(\cdot)$ or a collection of truly random functions $G(\cdot)$ or $P(\cdot)$. $\mathcal{B}$ simulates the environment for $\mathcal{A}$ as in Game 1, but whenever the simulation requires computing a PRF value (e.g., during token generation), $\mathcal{B}$ queries its oracle $\mathcal{O}$ for the relevant wallet on the relevant input and uses the response.

If $\mathcal{O}$ is a collection of DY PRFs, $\mathcal{B}$ creates the same view for $\mathcal{A}$ as in Game 1. This is because all other elements of the adversary's view are distributed independently of the seeds: C_{A} corresponding to a user's wallet is perfectly hiding (and is therefore independent of the seeds), and the proof π_{T} is is perfect zero-knowledge.

If $\mathcal{O}$ is a random function, $\mathcal{B}$ creates the same view for $\mathcal{A}$ as in Game 2: the serial numbers and the coefficients in the double-spending equation are perfectly random, and the proofs are simulated.

Therefore, if $\mathcal{A}$ distinguishes Game 1 from Game 2 with non-negligible advantage, $\mathcal{B}$ distinguishes an oracle that contains a collection of DY PRFs from an oracle that contains a collection of random functions with the same advantage, breaking the security of the DY PRF. [*End of proof of Claim 2.*]

Game 3. This game is exactly the same as Game 2, except that the double spending equation is replaced by the output of a truly random function. Thus instead of computing $D = g^{\mathsf{sk}_\mathsf{U}} * B^n$ as calculated in Game 2, where B is the output of a random function, the whole equation is replaced by the output of a random function, so D is truly random.

Claim 3. *The adversary's view in Game 2 is the same as in Game 3 because they are identical.*

Proof. In Game 2, the value B is the output of a random function (simulating the DY PRF). B is uniformly distributed in the target group $\mathbb{G}$, and multiplication by a fixed group element $g^{\mathsf{sk}_\mathsf{U}}$ is a permutation over $\mathbb{G}$, so the resulting value $D = g^{\mathsf{sk}_\mathsf{U}} \cdot B^n$ is also uniformly distributed in $\mathbb{G}$. [*End of proof of Claim 3.*]

Game 4. This game is identical to Game 3, except for the message used in the token generation oracle $\mathcal{O}_{\mathsf{GenToken}}$. In Game 3, the token was generated using user i's real message that includes the final commitment of attributes, the random group element R and encrypted metadata bit E_i as $m_i = (C_{\mathsf{F}_i}, R_i, E_i)$. Then the token included the values S and D that were outputs of a random function, along with the simulated proof π_T. In Game 4, the token generation is completely simulated using $\mathcal{S}_{\mathsf{GenToken}}$ as defined above. The simulated token is outputted as $T = (\sigma_1', m_1', S, D, \pi_\mathsf{T})$.

Claim 4. *A PPT adversary cannot distinguish Game 3 from Game 4, except with negligible probability.*

We prove Claim 4 with a hybrid argument. Let H_i be a hybrid game where the first i calls to the token generation oracle $\mathcal{O}_{\mathsf{GenToken}}$ are done by the simulator which calls its own token generation algorithm $\mathcal{S}_{\mathsf{GenToken}}$. In this notation, H_0 corresponds to Game 3 (all real messages) and H_q corresponds to Game 4 (all fixed messages), where q is the total number of queries.

We construct a reduction $\mathcal{B}$ that breaks the perfect adaptation property of the SEQ signature scheme given an adversary $\mathcal{A}$ that distinguishes between H_{i-1} and H_i with non-negligible probability. We rely on the security SEQ signatures described in Section 3.

Proof (Proof of Claim 4). The reduction $\mathcal{B}$ proceeds as follows:

$\mathcal{B}$ receives the public parameters pp from its challenger $\mathcal{C}_{\mathsf{SEQ}}$. $\mathcal{C}$ also sends over it's challenge (μ_0, μ), to $\mathcal{B}$. $\mu_0 = (C_0, R_0)$ is a pair of random group elements; $\mu = (C', R')$ is either equivalent to μ_0 or also a pair of group elements. It is $\mathcal{B}$'s challenge to guess whether μ is the randomization of μ_0. $\mathcal{B}$ proceeds by forwarding the public parameters (pp) to $\mathcal{A}_1$. Then, $\mathcal{A}_1$ generates the keys for

the Issuer and Verifier and sends the public key $\mathsf{pk_I}$ and $\mathsf{pk_V}$ and a proof of knowledge π_V and π_I of correct computation for both keys to $\mathcal{B}$. Now, $\mathcal{B}$ runs the extractor on the proof to extract the issuer's secret key $\mathsf{sk_I}$, and the verifier's secret key $\mathsf{sk_V}$. $\mathcal{B}$ also picks at random an index j: the reduction will obtain helpful information from $\mathcal{A}$ whenever the j^{th} query to $\mathcal{O}_{\mathsf{Req}}$ corresponds to the wallet from which token produced by the i^{th} query to $\mathcal{O}_{\mathsf{GenToken}}$.

For every k-th query, $\mathcal{B}$ simulates the oracles for $\mathcal{A}_2$ as follows:

- $\mathcal{O}_{\mathsf{UKGen}}$: $\mathcal{B}$ responds honestly and generates users using the algorithm defined above for $\mathcal{S}_{\mathsf{KeyGen_U}}$ which is identical to that of honest key generation $\mathsf{KeyGen_U}$.
- $\mathcal{O}_{\mathsf{Req}}$: For the k^{th} query, $\mathcal{B}$ will respond with a req, as follows. $\mathcal{B}$ picks it's seeds $z_{\mathsf{U0}}, z_{\mathsf{U1}}$ used later for the DY PRF keys in $\mathsf{GenToken}$. Note that $z_{\mathsf{U0}}, z_{\mathsf{U1}}$ are the outputs of a the hash function H and computed as $H(\mathsf{req}, 1)$ and $H(\mathsf{req}, 2)$. For $k = j$, $\mathcal{B}$ uses $\mu_0 = (C_0, R_0)$ as follows: $\mathcal{B}$ generates the commitment C_A sent in the request using C_0 by solving for $C_\mathsf{A} = C_0 / y_0^{z_{\mathsf{U0}}} * y_1^{z_{\mathsf{U1}}}$, then simulates the proof π_A and sends the request along with the state $(\mathsf{state}, \mathsf{req}) = (\mathsf{pk}_{\mathsf{U}_1}, C_\mathsf{A}, \pi_\mathsf{A})$.
  For all other queries, the reduction responds honestly as in the $\mathsf{Request}$ algorithm defined in the Cheshire construction.
- $\mathcal{O}_{\mathsf{GenToken}}$: After receiving requests $\mathcal{A}_2$ forms credentials cred and then for the k-th query requests tokens from these credentials corresponding to the correct request. If $\mathcal{A}_2$ asks for a token for an invalid cred or if the credential corresponding to cred has exceeded the token limit for N, then the reduction outputs $\perp$. If $k \neq i$: $\mathcal{B}$ generates the token exactly as described in $\mathcal{S}_{\mathsf{GenToken}}$ (if $k < i$) or follows the rules in Game 3 (if $k > i$).
  If $k = i$, then $\mathcal{B}$ uses the challenge message from $\mathcal{C}$, which is $\mu = (C', R')$, to form the token. First, $\mathcal{B}$ will compute S and D as in Game 3. $\mathcal{B}$ forms the "ciphertext" C by raising the base R' from μ to the appropriate power x_b. Then, $\mathcal{B}$ forms the message for the token as follows: $m = (g_1, C', R', E)$. Next, $\mathcal{B}$ uses the secret key of the issuer $\mathsf{sk_I}$ it extracted to sign the message to compute the signature σ. Then $\mathcal{B}$ simulates the proof π_T. Note that, in the event that μ_0 and μ are equivalent, this signature is distributed exactly as if it were adapted from μ_0, ensuring that the token is distributed as in Hybrid i. $\mathcal{B}$ outputs the token $T = (\sigma, m, S, D, \pi_\mathsf{T})$.

Eventually, $\mathcal{A}_2$ outputs a guess as to whether the i^{th} token in $\mathcal{O}_{\mathsf{GenToken}}$ was real or simulated. If $\mathcal{A}_2$'s guess was that it was simulated then $\mathcal{B}$ will output "not equivalent," otherwise "equivalent." If $\mathcal{A}$ succeeds with non-negligible advantage, $\mathcal{B}$ breaks the perfect adaptation property of the SEQ signature scheme. [*End of proof of Claim 4.*]

Game 5. All $\mathcal{O}_{\mathsf{Req}}$ outputs are simulated req's. This means that the commitment C_A is not a true commitment for the selected attributes, A. Instead C_A is sampled at random from the group. Note that in this experiment, the adversary's view is as in the unlinkability experiment $\mathsf{Exp}^{\mathsf{unlink}}_{\mathsf{SMA^2RT},\mathcal{A},\mathcal{S}}(\lambda)$ with $\beta = 1$.

Claim 5. *A PPT adversary cannot distinguish Game 4 from Game 5, except with negligible probability.*

The proof follows from the hiding property of the multi-base blindable Pedersen commitment. In Game 5, the reduction simulates the commitment in the request rather than computing them honestly. Since Pedersen commitments are uniformly chosen group elements, the resulting distribution is identical.

This concludes the sequence of games. Since Game 0 and Game 5 are indistinguishable, the scheme satisfies unlinkability.

□

Remark 5.4. We can also show that we satisfy unlinkability with double-spending. When double-spending a token $T = (\sigma', m', S, D, \pi_T)$ on behalf of the user with public key pk, the simulator solves for the value $B = (D/\mathsf{pk})^{1/H(\mathsf{ctx})}$, and then plugs it in to compute the value D' for the double-spent token T'. The rest of the proof stays the same.

Theorem 5.5 (Exculpability). *If the DY construction is a PRF, then the Cheshire construction satisfies the* $\mathsf{SMA}^2\mathsf{RT}$ *exculpability (Definition 2.7).*

Proof (Sketch of Theorem 5.5). We prove this theorem using a sequence of games that transforms the adversary's advantage in the real experiment into one where a successful adversary implies a solution to the discrete logarithm (DL) problem.

Game 0. Real experiment $\mathsf{Exp}^{\mathsf{exculp}}_{\mathsf{SMA}^2\mathsf{RT}}(\mathcal{A}, \lambda)$. The adversary acts as a malicious issuer interacting with honest users. To win, the adversary must output two tokens (T_i, T_j), with the same serial number $S_i = S_j$, and different contexts $\mathsf{ctx}_i \neq \mathsf{ctx}_j$, that frame an honest user.

↕ *Claim 1: zero-knowledge property*

Game 1. Identical to Game 0, except that all NIZK proofs generated by honest users in $\mathcal{O}_{\mathsf{Req}}$ and $\mathcal{O}_{\mathsf{GenToken}}$ are replaced by simulated proofs.

↕ *Claim 2: security of the PRF*

Game 2. Identical to Game 1, except that the DY PRF (used to calculate the serial numbers S and blinding factors B) is replaced by truly random functions.

↕ *Claim 3: knowledge-extractor properties and DL assumption*

Game 3. We now apply a knowledge extractor $\mathcal{E}$ to the adversary's forged token T_i (the token not produced by an honest oracle). If the adversary wins with non-negligible probability, the extractor successfully retrieves the witness from the proof π_{T_i}. We analyze two cases:

- Case 1: The adversary creates another token that reuses a serial number S from a previous honest token. By extracting the witness (specifically the PRF seed z_0), the reduction can solve the discrete logarithm problem.
- Case 2: The adversary outputs two new tokens with the same serial number on behalf of an honest user's keys. By extracting the witness (specifically the user's secret key sk_{U}), the reduction can solve the discrete logarithm problem.

In both cases, a successful adversary allows the reduction to break the discrete logarithm assumption. Therefore, the adversary's advantage must be negligible. □

Proof (Full Proof of Theorem 5.5).

Game 0. This is the real experiment $\mathsf{Exp}^{\mathsf{exculp}}_{\mathsf{SMA^2RT}}(\mathcal{A}, \lambda)$.

The adversary acts as a malicious Issuer and interacts with honest user instances via the $\mathcal{O}_{\mathsf{UKGen}}$, $\mathcal{O}_{\mathsf{Req}}$, and $\mathcal{O}_{\mathsf{GenToken}}$ oracles. The challenger honestly generates users' keys in $\mathcal{O}_{\mathsf{UKGen}}$ and in $\mathcal{O}_{\mathsf{Req}}$ ensures that the user the Issuer is requesting is an honest user, and then calls the $\mathsf{Request}$ algorithm honestly as in the Cheshire construction. The queries to $\mathcal{O}_{\mathsf{GenToken}}$ are also formed honestly as in the $\mathsf{GenToken}$ algorithm in the Cheshire construction. To win the game, the adversary must output at least two tokens that frame a user. Assume that $T_i = (\sigma'_i, m'_i, S_i, D_i, \pi_{\mathsf{T}_i})$ and $T_j = (\sigma'_j, m'_j, S_j, D_j, \pi_{\mathsf{T}_j})$ are the two tokens outputted by the adversary that successfully trigger the winning condition (i.e. $S_i = S_j$ and $\mathsf{ctx}_i \neq \mathsf{ctx}_j$), and let T_i be the token outputted by the adversary that was never produced by an honest user. The adversary can win in two possible ways. In case (1), the token T_j was produced by an honest user from the oracle, $\mathcal{O}_{\mathsf{GenToken}}$. In case (2), the token T_j is a new token created by the adversary.

Game 1. In this game, we simulate the proofs generated by the honest user.

The challenger replaces the real NIZK proofs in all responses from $\mathcal{O}_{\mathsf{Req}}$ and $\mathcal{O}_{\mathsf{GenToken}}$ with simulated proofs using the NIZK Simulator. This includes the zero knowledge proof π_{A} that the user creates in $\mathcal{O}_{\mathsf{Req}}$ to prove honest commitment to its PRF seeds, attributes, and its secret key. It also includes the zero knowledge proof π_{T} in $\mathcal{O}_{\mathsf{GenToken}}$ where the user proves it has correctly used its PRF seeds to compute the serial number, double spending identification, along with honest commitment, and tokens being within the range N.

Claim 1: *A PPT adversary cannot distinguish Game 0 from Game 1, except with negligible probability.*

The only difference between the two games is zero-knowledge simulation. In Game 0, the challenger generates real proofs in $\mathcal{O}_{\mathsf{Req}}$ where it outputs π_{A} to prove the attributes were honestly committed in C_{A}. In $\mathcal{O}_{\mathsf{GenToken}}$ the challenger produces a proof π_{T} to prove correct computation of the serial number, double spending equation, along with correctly committed values, predicate satisfaction, and proves the counter is within range. In contrast, the simulator will simulate the proofs π_{A} and π_{T} in Game 1. Since these are Fiat-Shamir-based proofs of discrete-log relations in prime-order groups in the random-oracle model, the distribution of the real proofs in Game 0 is identical to the simulated proofs in Game 1.

Game 2. In this game, all instances of the PRF evaluation are replaced by a truly random function.

Specifically, when the oracle $\mathcal{O}_{\mathsf{GenToken}}$ needs to evaluate the serial number $S = F^{\mathsf{DY}}_{z_0}(\mathsf{ctr})$ and the blinding factor $B = F^{\mathsf{DY}}_{z_1}(\mathsf{ctr})$, it will replace them with a (unique to this wallet) truly random functions, G and P from the PRF range, such that $S = G(\mathsf{ctr})$ and $B = P(\mathsf{ctr})$. $\mathcal{C}$ will compute these random functions as follows: it maintains a lookup table of inputs and outputs to both random

functions. If an input has been queried before, it retrieves the previous output from the table; otherwise, it samples a new random value from the DY PRF range and adds this new input-output pair to the table.

Claim 2. *A PPT adversary cannot distinguish Game 1 from Game 2, except with negligible probability.*

The indistinguishability of Game 1 and Game 2 follows directly from the pseudorandomness property of the DY PRF. We construct a reduction $\mathcal{B}$ that breaks the PRF security given a distinguisher $\mathcal{A}$ for the games.

Proof (Proof of Claim 2). First, note that, by a standard hybrid argument, it is sufficient to show a reduction that distinguishes an oracle that contains a collection of random functions (two functions, G and P for each wallet) from an oracle that contains a collection of DY PRFs: two per wallet.

The reduction $\mathcal{B}$ interacts with a the challenger that provides access to the oracle $\mathcal{O}$, which is either a collection of DY PRF functions $F_k^{\mathsf{DY}}(\cdot)$ or a collection of truly random functions $G(\cdot)$ or $P(\cdot)$. $\mathcal{B}$ simulates the environment for $\mathcal{A}$ as in Game 1, but whenever the simulation requires computing a PRF value (e.g., during token generation), $\mathcal{B}$ queries its oracle $\mathcal{O}$ for the relevant wallet on the relevant input and uses the response.

If $\mathcal{O}$ is a collection of DY PRFs, $\mathcal{B}$ creates the same view for $\mathcal{A}$ as in Game 1. This is because all other elements of the adversary's view are distributed independently of the seeds: C_{A} corresponding to a user's wallet is perfectly hiding (and is therefore independent of the seeds), and the proof π_{T} is is perfect zero-knowledge.

If $\mathcal{O}$ is a random function, $\mathcal{B}$ creates the same view for $\mathcal{A}$ as in Game 2: the serial numbers and the coefficients in the double-spending equation are perfectly random, and the proofs are simulated.

Therefore, if $\mathcal{A}$ distinguishes Game 1 from Game 2 with non-negligible advantage, $\mathcal{B}$ distinguishes an oracle that contains a collection of DY PRFs from an oracle that contains a collection of random functions with the same advantage, breaking the security of the DY PRF. [*End of proof of Claim 2.*]

Game 3. In this game, we run the adversary as in Game 2, and then after it terminates, run the knowledge extractor with black-box access to the adversary in order to obtain the opening to the commitment $C_{\mathsf{F}} = g_0^{z0}...h^r$ in the token T_i that is the new token that triggers the adversary's winning condition. More precisely:

Let $T_i = (\sigma'_i, m'_i, S_i, D_i, \pi_{\mathsf{T}_i})$ and $T_j = (\sigma'_j, m'_j, S_j, D_j, \pi_{\mathsf{T}_j})$ be the two tokens outputted by the adversary that successfully trigger the winning condition (i.e., they share a serial number such that $S_i = S_j$ and they identify an honest user $\mathsf{pk}_{\mathsf{U}} \in \mathcal{U}$). This means that the $\mathsf{IdentifyUser}$ algorithm must output the public key of an honest user in $\mathcal{U}$. Either the adversary outputs a tokens whose serial number is the same as a serial number of an honest user, or these tokens have a serial number that is new.

In either case, one of these tokens must be a token that was not generated by an honest user. Let T_i be the token outputted by the adversary that was

never produced by an honest user. The challenger runs the NIZK Knowledge Extractor $\mathcal{E}$ on the proof π_{T_i} contained within the forged token T_i.

In this game, the challenger rewinds the adversary to the step where it issued the random-oracle call that led it to eventually output π_{T}; the experiment produces a different response from the random oracle, and runs $\mathcal{A}$ to completion; if in this run $\mathcal{A}$ did not produce a proof π'_{T} that allows for the Σ-protocol's extraction of the witness $C_{\mathsf{F}} = y_{\ell-2}^{z0}...h^r$, then the game rewinds the adversary and tries again. Eventually, with overwhelming probability, the modified experiment will obtain the witness $C_{\mathsf{F}} = y_{\ell-2}^{z0}...h^r$, which proves the following claim:

Claim 3a: *If the adversary wins in Game 2 with non-negligible probability, then the knowledge extractor in Game 3 will succeed in computing an opening to* C_{F}.

We can use this knowledge extractor in order to contradict the discrete logarithm assumption.

Claim 3b: *If the adversary wins in Game 3 with non-negligible probability, this breaks the discrete logarithm assumption.*

We divide this claim into the two winning cases and provide reductions for each.

We first prove case 1 (when the adversary's forged token reuses an honest token's serial number) via a reduction $\mathcal{B}$ that breaks the discrete logarithm (DL) assumption given an adversary $\mathcal{A}$ that can produce another token T_i, which frames the user against one of the user's tokens T_j so that $S_i = S_j$.

Proof (Proof of Claim 3a). The reduction $\mathcal{B}$ interacts with the DL challenger who first generates the public parameters $(\mathbb{G}, p, g)$, samples a random value $x \xleftarrow{\$} \mathbb{Z}_p$, and sets $h = g^x$. The challenger forwards $(\mathbb{G}, p, g, h)$ to the reduction $\mathcal{B}$. The reduction $\mathcal{B}$ must answer $\mathcal{A}$'s queries to the $\mathcal{O}_{\mathsf{UKGen}}, \mathcal{O}_{\mathsf{Req}}$ and $\mathcal{O}_{\mathsf{GenToken}}$. It does so as described in Game 2, except that in $\mathcal{O}_{\mathsf{GenToken}}$ for each query k when creating the serial numbers, it samples a random value $r_k \xleftarrow{\$} \mathbb{Z}_p$ and sets the serial number as $S_k = h^{r_k}$.

Eventually, $\mathcal{A}$ will output two tokens T_i and T_j such that T_i was never outputted by $\mathcal{B}$, but T_j was outputted by $\mathcal{B}$ and the serial numbers of both tokens are equal: $S_i = S_j$. Then $\mathcal{B}$ uses the extractor $\mathcal{E}$ to extract a valid witness w_i from π_{T_i}. The witness w_i must include the openings to the commitment vector, specifically the secret seed z_{0_i} used to derive the serial number S_i. The knowledge extraction property implies that $S_i = F_{z_{0_i}}^{\mathsf{DY}}(\mathsf{ctr})$. Recall, that $\mathcal{B}$ calculated the serial number S_i by using its DL challenge h, so $S_i = h^{r_i}$. And by the calculation of the DY PRF, this means that the adversary $\mathcal{A}$ calculated $S_i = h^{r_i} = g^{\frac{1}{\mathsf{ctr}+z_{0_i}}}$. Therefore, $\mathcal{B}$ uses the seed z_{0_i} extracted from the witness to solve for the DL of h because $h = g^{\frac{1}{r_i(z_{0_i}+\mathsf{ctr})}}$. Thus, $\mathcal{B}$ will output $\frac{1}{r_i(z_{0_i}+\mathsf{ctr})}$ to its challenger, breaking the discrete log assumption.

Since $\mathcal{A}$ is able to output a token T_i to frame the user in case 1 with non-negligible probability, this implies that $\mathcal{B}$ is able to break the discrete logarithm with at least $\mathcal{A}$'s probability of winning case 1. [*End of proof of Claim 3a.*]

We now prove case 2 (the forged tokens have a heretofore unused serial number) via a reduction $\mathcal{B}$ that breaks the discrete logarithm (DL) assumption given an adversary $\mathcal{A}$ that can produce two new tokens T_i and T_j with the same serial numbers $S_i = S_j$ that were never outputted by an honest user, but are able to frame an honest user.

Proof (Proof of Claim 3b). The reduction $\mathcal{B}$ interacts with the DL challenger that first generates the public parameters $(\mathbb{G}, p, g)$, samples a random value $x \xleftarrow{\$} \mathbb{Z}_p$, and sets $h = g^x$. The challenger forwards $(\mathbb{G}, p, g, h)$ to the reduction $\mathcal{B}$. The reduction $\mathcal{B}$ must answer $\mathcal{A}$'s queries to the $\mathcal{O}_{\mathsf{UKGen}}, \mathcal{O}_{\mathsf{Req}}$ and $\mathcal{O}_{\mathsf{GenToken}}$. It does so as described in Game 2, except that in $\mathcal{O}_{\mathsf{UKGen}}$ for each user's key generation, it samples a random value $r_i \xleftarrow{\$} \mathbb{Z}_p$ and sets the public key of the user as $\mathsf{pk}_{\mathsf{U}} = h^{r_i}$. Any queries to the oracles $\mathcal{O}_{\mathsf{Req}}$ and $\mathcal{O}_{\mathsf{GenToken}}$ by $\mathcal{A}$ are answered honestly. Since all proofs are simulated, as in Game 2, the reduction does not need to know the users' secret keys when answering any of $\mathcal{A}$'s requests.

Eventually, $\mathcal{A}$ will output two tokens T_i and T_j such that both of them were never outputted by a query to $\mathcal{B}$ and the serial numbers of both tokens are equal such that $S_i = S_j$. The resulting output of the $\mathsf{IdentifyUser}$ function outputs the public key of an honest user pk_{U}. $\mathcal{B}$ uses the extractor $\mathcal{E}$ to extract a valid witness w_i from π_{T_i}. The witness w_i must include the openings to the commitment vector, specifically the secret key of the user sk_{U}. The knowledge extraction property of the proof system implies that $\mathsf{pk}_{\mathsf{U}} = g^{\mathsf{sk}_{\mathsf{U}}}$. Recall, that $\mathcal{B}$ outputted every honest user's public keys as $\mathsf{pk}_{\mathsf{U}} = h^{r_i}$ for some randomly chosen r_i for each user. Since $\mathcal{B}$ was able to extract the secret key sk_{U} from the witness, it means that the secret key must be xr_i since $\mathsf{pk}_{\mathsf{U}} = g^{\mathsf{sk}_{\mathsf{U}}} = h^{r_i} = g^{xr_i}$. $\mathcal{B}$ solves for x by dividing xr_i by r_i, and returns x to its DL challenger.

Since $\mathcal{A}$ is able to output two tokens T_i and T_j to frame a user in case 2 with non-negligible probability, this implies that $\mathcal{B}$ is able to break the discrete logarithm with at least $\mathcal{A}$'s probability of winning case 2. [*End of proof of Case 3b.*]

This concludes the series of games. Since Game 0 and Game 3 are indistinguishable, the scheme satisfies exculpability.

□

6 Acknowledgments

This research was supported by NSF awards 2312241 and 2247305. We thank the anonymous reviewers for their careful feedback, especially on the subtle points of the privacy of the metadata bit definition. We are grateful to Foteini Baldimtsi for helpful discussions.

References

1. Regulation (eu) 2024/1183 of the european parliament and of the council of 11 april 2024 amending regulation (eu) no 910/2014 as regards establishing the european digital identity framework (2024), https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32024R1183
2. Architecture documentation for the german national EUDI wallet (2025), https://bmi.usercontent.opencode.de/eudi-wallet/wallet-development-documentation-public
3. European Digital Identity Wallet. Architecture and Reference Framework. Version 2.4.0 (2025), https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/2.4.0/
4. 2025 (NL Wallet Software System overview), https://minbzk.github.io/nl-wallet/main/architecture/c4/software-system.html
5. Abdalla, M., Namprempre, C., Neven, G.: On the (im)possibility of blind message authentication codes. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 262–279. Springer, Berlin, Heidelberg (Feb 2006). https://doi.org/10.1007/11605805_17
6. Baldimtsi, F., Hanzlik, L., Nguyen, Q., Yadav, A.: Non-interactive anonymous tokens with private metadata bit. Cryptology ePrint Archive, Report 2025/430 (2025), https://eprint.iacr.org/2025/430
7. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 1087–1098. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516687
8. Baum, C., Blazy, O., Camenisch, J., Hoepman, J.H., Lee, E., Lehmann, A., Lysyanskaya, A., Mayrhofer, R., Montgomery, H., Nguyen, N.K., Preneel, B., abhi shelat, Slamanig, D., Tessaro, S., Thomsen, S.E., Troncoso, C.: Cryptographers' feedback on the eu digital identity's arf. https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/discussions/211 (June 2024), available online at https://github.com/user-attachments/files/15904122/cryptographers-feedback.pdf
9. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: Compact e-cash and simulatable VRFs revisited. In: Pairing-Based Cryptography - Pairing 2009, Third International Conference, Proceedings. Lecture Notes in Computer Science, vol. 5671, pp. 114–131. Springer (2009)
10. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006. pp. 390–399. ACM Press (Oct / Nov 2006). https://doi.org/10.1145/1180405.1180453
11. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168596
12. Benhamouda, F., Lepoint, T., Orrù, M., Raykova, M.: Publicly verifiable anonymous tokens with private metadata bit. Cryptology ePrint Archive, Report 2022/004 (2022), https://eprint.iacr.org/2022/004
13. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer, Berlin, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4_38

14. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC). pp. 103–112. ACM (1988)
15. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Berlin, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_4
16. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Berlin, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_31
17. Brands, S.: Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy. Ph.D. thesis, Eindhoven Inst. of Tech. The Netherlands (1999)
18. Camenisch, J., Chaabouni, R., shelat, a.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Berlin, Heidelberg (Dec 2008). https://doi.org/10.1007/978-3-540-89255-7_15
19. Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., Meyerovich, M.: How to win the clonewars: Efficient periodic n-times anonymous authentication. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006. pp. 201–210. ACM Press (Oct / Nov 2006). https://doi.org/10.1145/1180405.1180431
20. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Berlin, Heidelberg (May 2005). https://doi.org/10.1007/11426639_18
21. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Balancing accountability and privacy using e-cash (extended abstract). In: De Prisco, R., Yung, M. (eds.) SCN 06. LNCS, vol. 4116, pp. 141–155. Springer, Berlin, Heidelberg (Sep 2006). https://doi.org/10.1007/11832072_10
22. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Berlin, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_7
23. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 268–289. Springer, Berlin, Heidelberg (Sep 2003). https://doi.org/10.1007/3-540-36413-7_20
24. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28628-8_4
25. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski, Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 410–424. Springer, Berlin, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052252
26. Camenisch, J.L.: Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem. Ph.D. thesis, ETH Zürich (1998)
27. Chaabouni, R., Lipmaa, H., shelat, a.: Additive combinatorics and discrete logarithm based range protocols. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 10. LNCS, vol. 6168, pp. 336–351. Springer, Berlin, Heidelberg (Jul 2010). https://doi.org/10.1007/978-3-642-14081-5_21

28. Chairattana-Apirom, R., Döttling, N., Lysyanskaya, A., Tessaro, S.: Everlasting anonymous rate-limited tokens. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part VI. LNCS, vol. 16250, pp. 435–468. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5119-4_14
29. Chairattana-Apirom, R., Harding, F., Lysyanskaya, A., Tessaro, S.: Server-aided anonymous credentials. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VI. LNCS, vol. 16005, pp. 291–324. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01887-8_10
30. Chase, M., Durak, F.B., Vaudenay, S.: Anonymous tokens with stronger metadata bit hiding from algebraic MACs. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 418–449. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38545-2_14
31. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO'82. pp. 199–203. Plenum Press, New York, USA (1982). https://doi.org/10.1007/978-1-4757-0602-4_18
32. Cramer, R.: Modular Design of Secure yet Practical Cryptographic Protocol. Ph.D. thesis, University of Amsterdam (1997)
33. Cramer, R., Damgård, I.: Zero-knowledge proof for finite field arithmetic, or: Can zero-knowledge be for free? In: Krawczyk, H. (ed.) CRYPTO '98. LNCS, vol. 1642, pp. 424–441. Springer Verlag, Berlin (1998)
34. Damgård, I.: On σ-protocols. Available at http://www.daimi.au.dk/~ivan/Sigma.ps (2002)
35. Davidson, A., Chalkias, F., Cong, L., Faz-Hernandez, A., Ni, M., Paquin, C.: Trust tokens: Privacy-preserving proof of authenticity. Cryptology ePrint Archive, Report 2020/072 (2020), https://ia.cr/2020/072
36. Davidson, A., Goldberg, I., Sullivan, N., Tankersley, G., Valsorda, F.: Privacy pass: Bypassing internet challenges anonymously. In: Proceedings on Privacy Enhancing Technologies. vol. 2018, pp. 164–180 (2018)
37. Desmoulins, N., Dumanois, A., Kane, S., Traoré, J.: Making BBS anonymous credentials eIDAS 2.0 compliant. Cryptology ePrint Archive, Paper 2025/619 (2025), https://eprint.iacr.org/2025/619
38. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Berlin, Heidelberg (Jan 2005). https://doi.org/10.1007/978-3-540-30580-4_28
39. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Berlin, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34931-7_5
40. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Berlin, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
41. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Berlin, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_10
42. Friedrichs, K., Harding, F., Lehmann, A., Lysyanskaya, A.: Device-bound anonymous credentials with(out) trusted hardware. Cryptology ePrint Archive, Report 2025/1995 (2025), https://eprint.iacr.org/2025/1995
43. Fuchsbauer, G., Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. Journal of Cryptology **32**(2), 498–546 (Apr 2019). https://doi.org/10.1007/s00145-018-9281-4

44. Juels, A., Luby, M., Ostrovsky, R.: Security of blind digital signatures (extended abstract). In: Kaliski, Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 150–164. Springer, Berlin, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052233
45. Kreuter, B., Lepoint, T., Orrù, M., Raykova, M.: Anonymous tokens with private metadata bit. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 308–336. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_11
46. Lipmaa, H.: Statistical zero-knowledge proofs from diophantine equations. Cryptology ePrint Archive, Report 2001/086 (2001), https://eprint.iacr.org/2001/086
47. Lysyanskaya, A.: Signature schemes and applications to cryptographic protocol design. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts (Sep 2002)
48. Lysyanskaya, A.: Security analysis of RSA-BSSA. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 251–280. Springer, Cham (May 2023). https://doi.org/10.1007/978-3-031-31368-4_10
49. Orru, M., Yun, C.: Interactive Sigma Proofs. Internet-Draft draft-irtf-cfrg-sigma-protocols-01, Internet Engineering Task Force, https://datatracker.ietf.org/doc/draft-irtf-cfrg-sigma-protocols/, work in Progress
50. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 387–398. Springer, Berlin, Heidelberg (May 1996). https://doi.org/10.1007/3-540-68339-9_33
51. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3), 361–396 (Jun 2000). https://doi.org/10.1007/s001450010003
52. Rannenberg, K., Camenisch, J., Sabouri, A. (eds.): Attribute-based Credentials for Trust: Identity in the Information Society. Springer (2015). https://doi.org/10.1007/978-3-319-14439-9, https://doi.org/10.1007/978-3-319-14439-9
53. Schröder, D., Unruh, D.: Security of blind signatures revisited. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 662–679. Springer, Berlin, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-30057-8_39
54. Thomson, M., Wood, C.A.: Oblivious HTTP (Jan 2024), https://www.ietf.org/rfc/rfc9458.html
55. Yun, C., Wood, C.A.: Anonymous rate-limited credentials. Internet-Draft draft-yun-cfrg-arc-00, Internet Engineering Task Force (February 2025), https://datatracker.ietf.org/doc/draft-yun-cfrg-arc/, work in Progress
56. Yun, C., Wood, C.A.: Anonymous Rate-Limited Credentials. Internet-Draft draft-yun-cfrg-arc-01, Internet Engineering Task Force (Aug 2025), https://datatracker.ietf.org/doc/draft-yun-cfrg-arc/01/, work in Progress